UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO E SISTEMAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

LORÍ VIALI

SIMULAÇÃO APLICADA À MANUFATURA FLEXÍVEL

DISSERTAÇÃO SUBMETIDA À UNIVERSIDADE FEDERAL DE SANTA CATARINA
PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA

FLORIANÓPOLIS, OUTUBRO DE 1991
SC - BRASIL

SIMULAÇÃO APLICADA À MANUFATURA FLEXIVEL

Lorí Viali

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA OBTENÇÃO DO TÍTULO DE

MESTRE EM ENGENHARIA

ESPECIALIDADE ENGENHARIA DE PRODUÇÃO E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO

Prof. Neri dos Santos, Dr. Ing.
Coordenador

Banca Examinadora:

Prof. Cristiano José C. de A. Cunha, Dr. rer. pol.
Orientador/Presidente

Prof. Edgar Augusto Lanzer, PhD

Prof. Paulo R. Nascimento, M.Sc.

À Dolores Etzberger Viali

AGRADECIMENTOS

- À Pontifícia Universidade Católica do Rio Grande do Sul, pelo liberação com vencimentos.

- À Universidade Federal do Rio Grande do Sul, pela liberação com vencimentos.

- Ao professsor Cristino José C. De A. Cunha, pela orientação e apoio no desenvolvimento deste trabalho.

- Ao professor Paulo de Freitas Filho, pela co-orientação e auxílio bibliográfico.

- Ao colega José Carlos Flack, pela interação bibliográfica.

- Aos professores Edgar Lanzer e Paulo Nascimento pela participação na banca.

RESUMO

Os sistemas flexíveis de manufatura resultaram do desejo de alcançar a produtividade da produção de massa com a agilidade da fabricação por encomenda. Em virtude do grau de automação, da flexibilidade, do número elevado de diferentes produtos fabricados em pequenos lotes, estes sistemas apresentam uma complexidade maior do que os tradicionais.

A obtenção da eficiência produtiva destes sistemas envolve otimizar um grande número de variáveis decisórias e solucionar uma problemática variada que vai de problemas de longo prazo, como os de projeto, médio prazo, representado pelo planejamento, curto prazo, identificado pelos problemas de programação e, ainda, por problemas de curtíssimo prazo, relacionados com o controle em tempo-real do sistema.

A complexidade destes sistemas fez da simulação senão a única ferramenta para sua análise, pelo menos, a mais importante. Com o barateamento dos custos de hardware e a crescente aceitação da simulação como técnica de solução de problemas, o número de recursos de software relacionados aos SFM cresceu muito além do que qualquer um seria capaz de acompanhar.

O propósito deste trabalho é descrever estes sistemas, sua origem, vantagens e problemática, classificando e analisando os instrumentos de software disponíveis possibilitando, desta forma, decisões melhores e mais rápidas por parte de quem pretenda fazer uso da simulação para estudá-los.

ABSTRACT

The object to reach the mass production output with the job shop flexibility led the flexible manufacturing system (FMS) development. These systems show a major complexity as a result of the flexibility, the degree of the automation and of the great number of distinct products manufatured.

The obtention of the productive efficiency of these systems involves optimizing a great number of the decision variables and to solve diverse problems such as project (long time), planning (medium time), programming (short time) and control (very short time) problems.

The complexity of these systems made the simulation not only the unique tool for its analysis, but at least the most important. With the decrease costs of hardwareand the increasing acceptance of the simulation as a problem solving technique, the number of softwares resources related to FMS analysis expanded further than anyone is able to follow.

The purpose of this work is to describe these systems, its sources, advantages and problems, classifying and analysing the software tools availables, making in this way the best and fastest decisions for one who pretends to make use of simulation to study them.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

CAPÍTULO I

1. INTRODUÇÃO

1.1. ORIGEM DO TRABALHO

Houve um aumento dramático no uso da simulação para análise de sistemas e resolução de problemas [52, 126, 191, 216 e 219]. De todas as aplicações de simulação, 59% são na área de manufatura, concentrando-se a maior parte na manufatura flexível [67]. No início de 1960 era difícil encontrar um artigo em qualquer área que envolvesse uma aplicação da simulação. Nos últimos anos tal foi o seu crescimento em popularidade que o difícil, hoje, é encontrar uma conferência ou um periódico sem alguma aplicação ou artigo sobre simulação [106].

Analisando-se parte desta literatura extremamente vasta, observa-se que o leque de instrumentos de software utilizados é extremamente variado, abrangendo desde linguagens de propósitos gerais, tais como, Fortran e C até simuladores específicos. Os estudos de simulação envolvendo a manufatura flexível abrangem aspectos que vão desde modelos macros que objetivam fazer uma análise de capacidade do sistema global até modelos ao nível micro que se ocupam, por exemplo, em determinar o número de veículos dirigidos automaticamente necessários num determinado sistema.

Os sistemas flexíveis de manufatura são complexos e constituídos de vários subsistemas. A problemática relacionada com estes sistemas é ampla envolvendo desde problemas de longo prazo, como os de projeto, até problemas em tempo muito curto, como os de controle. Em virtude de constituirem uma tecnologia relativamente recente, e ainda não cristalizada, a confusão de conceitos e pontos de vistas sobre o que constitui um problema de planejamento, projeto, programação ou controle de um Sistema Flexível de Manufatura, aumenta a dificuldade de quem se dispuser a estudar aspectos ou mesmo todo o sistema. Alie-se a isto o grande número de ferramentas de software desenvolvidos para analisar estes sistemas e tem-se um quadro da complexidade da tarefa.

Isto tem sido causado pela grande complexidade dos

sistemas automatizados, custos computacionais decrescentes de micros e estações de trabalho, melhoria dos softwares de simulação que reduzem o tempo de desenvolvimento dos modelos e a disponibilidade de animação gráfica que tem resultado no entendimento e uso da simulação por engenheiros e administradores. Como conseqüência uma pessoa tentando selecionar um software de simulação, para sua organização ou para uma aplicação particular, vê-se frente a uma desconcertante variedade de escolhas em termos de capacidades técnicas, facilidade de uso, formas de abordagem e custo. A situação é exarcerbada pelas freqüentes mudanças, renomeações ou adições aos softwares existentes e por uma introdução regular de produtos de simulação inteiramente novos.

Uma pessoa não familiarizada pode literalmente gastar três ou mais meses para avaliar cuidadosamente softwares para uma aplicação particular [124]. Considerando-se estes aspectos, constatou-se a necessidade de um trabalho que classificasse e analisasse os problemas relacionados com os sistemas flexíveis de manufatura bem como as opções computacionais para estudá-los.

1.2. OBJETIVOS DO TRABALHO

A utilização da simulação para analisar os Sistemas Flexíveis de Manufatura é feita através de uma ampla variedade de recursos computacionais (softwares) e trata de uma problemática variada. O objetivo deste trabalho é fornecer uma visão abrangente destes recursos, classificando-os pela sofisticação, isto é, pela capacidade de liberar o usuário da tarefa de programação e dar suporte as outras tarefas de um estudo de simulação, bem como, analisar a problemática associada aos sistemas flexíveis de manufatura e de que maneira a simulação está sendo utilizada para encaminhar estes problemas. O objetivo é fornecer, ainda, uma introdução aos SFM, através de sua conceituação, difusão, origem, estrutura e vantagens.

1.3. LIMITAÇÕES DO TRABALHO

O trabalho se limitou a analisar as alternativas de software tradicionais de simulação discreta. Desta forma, não foram considerados softwares de simulação concorrente ou paralela.

Também não foi abordada a simulação inteligente, ou seja, o casamento da simulação com os sistemas especialistas. Não foram considerados, ainda, os demais aspectos de um estudo completo de simulação como: modelagem, validação e verificação e análise dos resultados. Obviamente, as partes excluídas vão muito além das citadas acima, uma vez que simulação é um campo muito amplo e o número de trabalhos na área nos últimos anos têm crescido exponencialmente.

1.4. ESTRUTURA DO TRABALHO

O trabalho foi estruturado em 10 capítulos, assim, além deste que apresenta a origem, objetivo, limitações e estrutura do trabalho tem-se ainda:

O capítulo 2 que trata da conceituação, caracterização, origem, importância, vantagens, difusão e dos elementos componentes dos sistemas flexíveis de manufatura.

O capítulo 3 que trata da problemática do SFM, abordando os problemas de projeto, planejamento, programação e controle.

O capítulo 4 que aborda a simulação no contexto das técnicas de Pesquisa Operacional utilizadas para estudar os sistemas flexíveis de manufatura e coloca a codificação no contexto de um estudo completo de simulação.

O capítulo 5 que classifica os recursos de software para simular um SFM e analisa o uso das linguagens de propósito geral.

O capítulo 6 que analisa as extensões das linguagens de propósito geral para simulação, denominados de conjuntos de sub-rotinas ou pacotes.

O capítulo 7 que apresenta as opções de softwares constituída pelas linguagens de simulação.

O capítulo 8 que analisa os pacotes orientados por dados, constituídos pelos modelos genéricos de manufatura e pelos modelos genéricos de sistemas flexíveis de manufatura.

O capítulo 9 que analisa as opções de software de mais alto nível formada pelos programas geradores, pré-processadores e ambientes de suporte.

O capítulo 10 que apresenta as tendências e perspectivas da simulação, pesquisas correntes e progressos, conclusão e pesquisas adicionais.

CAPÍTULO 2

2. SISTEMAS FLEXÍVEIS DE MANUFATURA (SFM)

2.1. CONCEITUAÇÃO

Embora muito se tenha escrito sobre manufatura flexível em periódicos especializados, nos últimos anos, o problema de uma definição aceitável ainda não foi solucionado, talvez em virtude do assunto ser relativamente recente, as interpretações ainda não tenham se cristalizado, acarretando a grande variabilidade no entendimento do que seja um Sistema Flexível de Manufatura (SFM). A tentativa de uma definição ficou ainda mais difícil em virtude das discussões atuais, sobre um assunto relacionado, a denominada 'manufatura integrada por computador' ou, mais propriamente conhecida, pela sigla CIM (*Computer Integrated Manufacturing*). Dentre os inúmeros conceitos existentes, um apresenta-se bastante apropriado, pois relaciona o CIM com o SFM.

CIM é uma rede unificada de sistemas computacionais executando e/ou controlando a totalidade das funções integradas de um sistema de manufatura.

Um SFM é um ambiente computacional intensivo em que as funções de manufatura, movimentação de materiais, gerenciamento de ferramentas, programação e controle estão integradas. Por esta razão, um SFM é realmente a implementação do CIM ao nível do chão de fábrica [180].

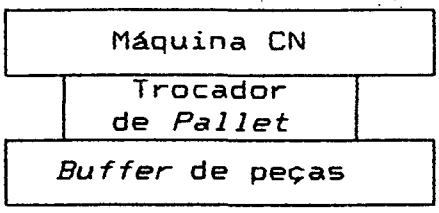
2.2. CARACTERIZAÇÃO

Baseado no número de máquinas de comando numérico (CN) e seus arranjos, um SFM pode ser classificado em [121]:

Módulo flexível de manufatura (MFM).

Um MFM pode ser definido como uma máquina CN aumentada por um *buffer* de peças, um trocador de ferramentas e um trocador de *pallets*. Os dois últimos ítens podem ser substituídos por um robô. A figura 2.1 ilustra o conceito de módulo flexível de manufatura.

Figura 2.1 - Módulo flexível de manufatura

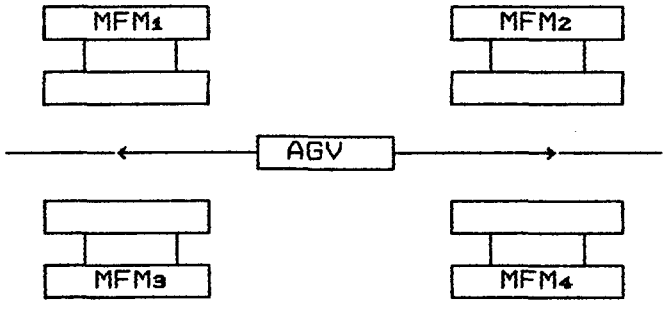


Fonte: **KUSIAK, 1985**

Célula flexível de manufatura (CFM)

Uma CFM consiste de vários MFM e pode ser construída de acordo com o tipo de produto ou processo. A figura 2.2 ilustra uma CFM com um veículo guiado automaticamente (AGV).

Figura 2.2 - Célula Flexível de Manufatura

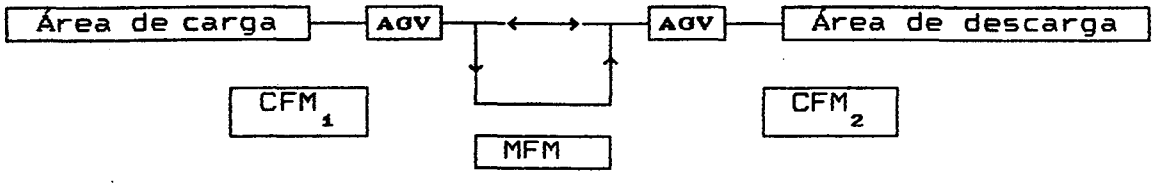


Fonte: **KUSIAK, 1985**

Grupo flexível de manufatura (GFM)

Um GFM é um conjunto de CFM e de MFM numa mesma área de manufatura ligados por um sistema de movimentação de materiais (SMM), normalmente um veículo dirigido automaticamente (AGV de *Automated Guided Veicle*), sob controle de um mesmo computador. A figura 2.3 ilustra um GFM.

Figura 2.3 - Grupo Flexível de Manufatura

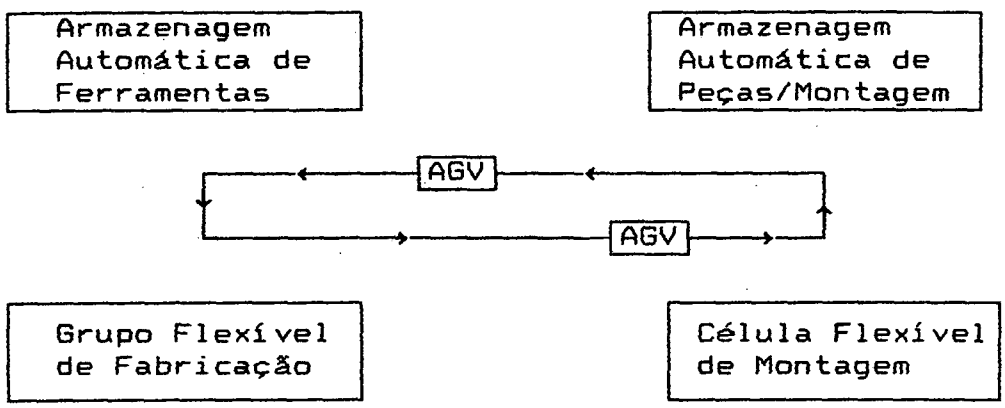


Fonte: **KUSIAK, 1985**

Sistema flexível de produção (SFP)

Um SFP consiste de CFM representando diferentes áreas de manufatura: fabricação, usinagem ou montagem. A figura 2.4 ilustra um SFP.

Figura 2.4 - Sistema Flexível de Produção

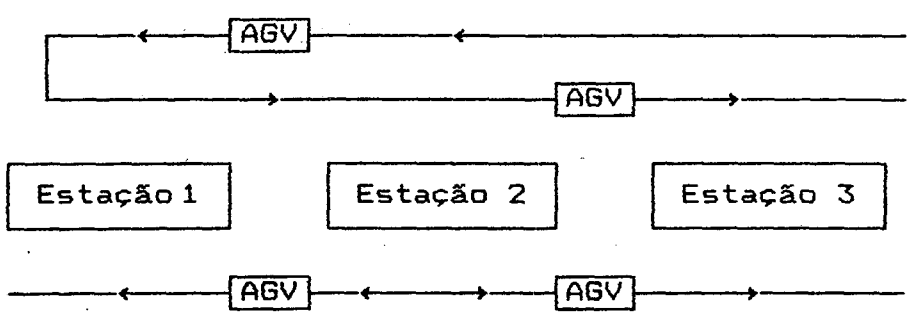


Fonte: **KUSIAK, 1986**

Linha flexível de manufatura (LFM)

Uma LFM pode ser definida como um conjunto de máquinas ferramentas dedicadas, que são denominadas de estações. Uma LFM pode ser de 5 tipos: (i) AGV, (ii) Robô, (iii) Esteira (iv) Vagonete (v) *Shuttle*. A figura 2.5 ilustra uma linha AGV.

Figura 2.5 - Linha Flexível de Manufatura

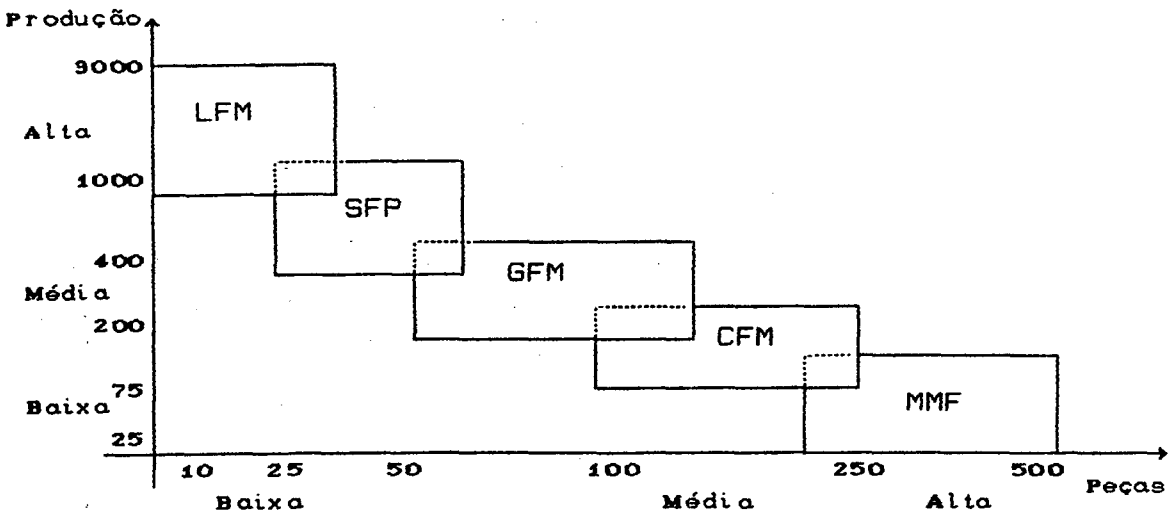


Fonte: **KUSIAK, 1985**

A figura 2.6 mostra a relação entre os tipos de SFM em função do número de componentes diferentes que podem produzir e da

capacidade de produção anual.

Figura 2.6 - Tipos de SFM x Produção x Número de peças diferentes



Fonte: KUSIAK, 1985

2.3. O SFM NO CONTEXTO DA MANUFATURA

Como forma de situar o SFM no contexto da manufatura, são colocados os ambientes tradicionais de produção, classificados de acordo com o tipo da planta e com o tipo de *layout* [91].

2.3.1. Plantas de produção

Em relação ao volume produtivo, as plantas de manufatura podem ser classificadas em produção *job shop*, de lote e de massa. Esta classificação é associada à manufatura de produtos discretos, mas também pode ser aplicada em indústrias de processos. Por exemplo, alguns químicos são produzidos em lotes (produção de lote), enquanto outros são produzidos por processos de fluxos contínuos (produção de massa).

1. *Job Shop*

A característica diferenciadora do *job shop* é o baixo volume. Os lotes manufaturados são pequenos, normalmente de um só tipo. A produção *job shop* é, geralmente, usada para satisfazer

necessidades específicas de consumidores. A diversidade de métodos de trabalho e plantas para fazê-lo é grande. Os equipamentos de produção são flexíveis e de propósitos gerais para permitir a variedade de trabalho. O nível de habilidade dos trabalhadores de *job shop* é relativamente alto, como forma de executarem diferentes atribuições. Exemplos de produtos manufaturados em um ambiente *job shop* são aviões, máquinas ferramentas e equipamentos especiais.

2. Produção de lote

Envolve a manufatura de lotes maiores do que num ambiente *job shop* de um mesmo ítem ou produto. Os equipamentos usados na produção de lotes são de propósito geral, porém projetados para taxas de produção mais altas. Por exemplo, um torno revólver, capaz de manter várias ferramentas de corte, é utilizado ao invés de um convencional. As máquinas ferramentas utilizadas são, muitas vezes, combinadas com moldes e fixadores especialmente projetados para aumentarem a taxa de saída.

Ítens feitos em ambientes do tipo lote incluem móveis, livros didáticos, componentes de muitos produtos de consumo montados como eletrodomésticos. Estima-se que cerca de 75% de todos os produtos que são manufaturados são produzidos em lotes de 50 ou menos. Caracteriza-se, assim, a importância da produção em lotes e do *job shop* na atividade manufatureira global [91].

3. Produção de massa.

Produção de massa é a manufatura especializada e contínua do mesmo produto. Caracteriza-se por altas taxas de produção, equipamentos e, às vezes, a planta totalmente dedicados a um produto. O equipamento é de propósito especial, ao invés de propósito geral. O investimento em máquinas e ferramental especializado é alto. O nível de perícia do trabalho em uma planta de produção de massa tende a ser menor do que em uma planta de lote ou *job shop*.

2.3.2. Tipos de *layouts* de plantas de produção

O termo '*layout* da planta' refere-se ao arranjo físico

dos equipamentos. Os *layouts* de planta, associados aos ambientes de manufatura tradicionais de produção, são os de posição fixa, de processo e de fluxo do produto, existindo uma correlação entre os tipos de *layout* da planta e o tipo de ambientes de produção.

1. *Layout* de posição fixa.

Neste *layout* o termo 'posição fixa' refere-se ao produto. Em virtude de seu tamanho e peso, o produto permanece em uma posição e o equipamento usado em sua fabricação e a mão de obra é que são deslocados até ele. Quando o trabalho é completado, o equipamento é removido. O produto é eventualmente movido para fora da planta, permanecendo a mesma para o próximo trabalho. Este tipo de arranjo é muitas vezes associado ao *job shop* em que produtos complexos são fabricados em quantidades muito pequenas.

2. *Layout* de processo.

No *layout* de processo, os equipamentos produtivos são arranjados em grupos de acordo com tipos gerais de processos de manufatura. Os tornos estão em um departamento, furadeiras em outro, moldagem em outro etc. A vantagem deste *layout* é sua flexibilidade. Peças diferentes, cada uma requerendo sua própria e única seqüência de operações, podem ser roteirizadas, através do respectivo departamento e numa ordem própria. O *layout* de processo é típico em produções *job shop* e de lote.

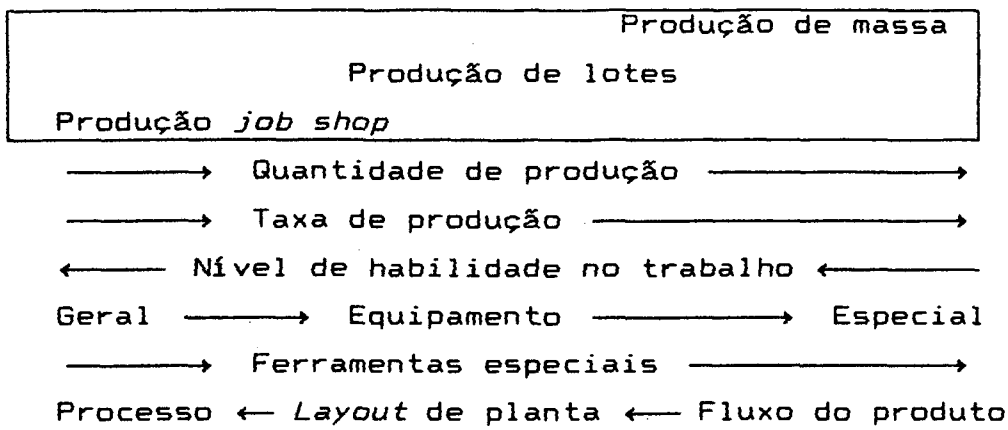
3. *Layout* fluxo do produto

Para produtos complexos montados ou ítens requerendo uma longa seqüência de operações, a eficiência produtiva é melhor obtida com o *layout* fluxo do produto. Neste tipo de *layout* os equipamentos de processamento são colocados ao longo da linha do fluxo do produto. O produto em processo é movimentado por esteiras ou meios equivalentes entre estações de trabalho sendo progressivamente fabricado. Este tipo de *layout* é apropriado para a produção de massa. O arranjo dos equipamentos dentro da planta é relativamente inflexível e é adequado somente quando as quantidades produzidas são grandes o suficiente para justificar o investimento.

A figura 2.7, ilustra os tipos de plantas de produção e

suas principais características diferenciadoras.

Figura 2.7 - Tipos de plantas de produção



Fonte: Groover, 1980

Estes três tipos de *layout* são os convencionais, encontrados em plantas de manufatura. Existe um quarto grupo, denominado de *layout* de grupo tecnológico, que representa uma tentativa para combinar a eficiência do *layout* de fluxo com a flexibilidade do *layout* do processo e é utilizado com os **sistemas flexíveis de manufatura.**

2.3.3 - Tecnologia de grupo

Tecnologia de grupo (TG) é um conjunto de técnicas que objetiva aumentar a produtividade de fabricantes de pequenos lotes. TG procura identificar famílias de peças que requerem processamento semelhante em um conjunto de máquinas e estas são, geralmente, agrupadas em células onde cada célula é capaz de satisfazer quase todas as necessidades de produção da família de peças atribuídas a ela. A formação destas células oferece o potencial para reduzir custos e tempos de *setup* [48].

A ênfase da literatura em TG tem gradualmente mudado, a partir da última década, do esquema de classificação para o problema de desenvolver métodos para agrupar peças e suas máquinas associadas. O esquema de classificação usa um sistema de códigos para descrever as características da peça, sua forma geométrica, o tipo de operação requerida e as suas seqüências. Existem vários sistemas de codificação em uso. Muitos deles utilizam vários dígitos para identificar uma peça, com cada dígito representando

um atributo. Um atributo representa a operação necessária para processar a peça, outro a máquina necessária, etc. Para identificar famílias de peças, métodos de agrupamento são, muitas vezes, utilizados. Algoritmos de agrupamento requerem uma medida de 'similaridade' entre estes códigos. Em muitos casos, a similaridade é medida pela distância entre 2 peças, que significa alguma espécie de média de diferenças entre os valores dos atributos dos dois códigos.

Uma deficiência séria deste método ocorre em casos na qual a codificação é subjetiva e o valor do código é um *mix* de dados nominais e numéricos com escalas escolhidas arbitrariamente. Neste caso, a 'distância' entre duas peças tem pequeno significado, pois o uso de diferentes sistemas de codificação resultará em distâncias drasticamente diferentes para o mesmo par de peças.

Uma vez que estes procedimentos não produzem resultados satisfatórios, a pesquisa em TG começou a procurar por métodos que agrupassem peças e máquinas associadas simultaneamente. Esta nova abordagem utiliza uma matriz máquina-peça, em lugar de longas cadeias de dígitos dos códigos. A matriz tem o elemento $a_{ij} = 1$, se a peça i deve ser processada na máquina j, e $a_{ij} = 0$ em caso contrário. A idéia básica é que a matriz possa ser re-arranjada de modo que os a_{ij} não nulos estejam todos agrupados diagonalmente, permitindo que a associação máquina-peça seja facilmente identificada.

Muitos algoritmos têm sido desenvolvidos para encontrar esta forma bloco diagonal. A principal desvantagem destes algoritmos, bem como de outras técnicas de TG é a hipótese de demanda temporal uniforme para cada peça. Quando uma máquina é necessária para processar muitas peças diferentes, técnicas de TG podem especificar múltiplas máquinas idênticas para atenuar o gargalo potencial. Entretanto o gargalo resultante de um repentino aumento na demanda de uma certa peça não é considerado.

TG tem provado ser útil para selecionar peças e máquinas no estágio de projeto do SFM [105]. Nesta fase, o objetivo é identificar que tipos de peças são apropriadas para a produção no SFM em termos de tamanho, número de fixadores necessários e tipos de operações de máquinas requeridas. Pela classificação de peças, utilizando sistema de codificação TG, pode-se identificar o

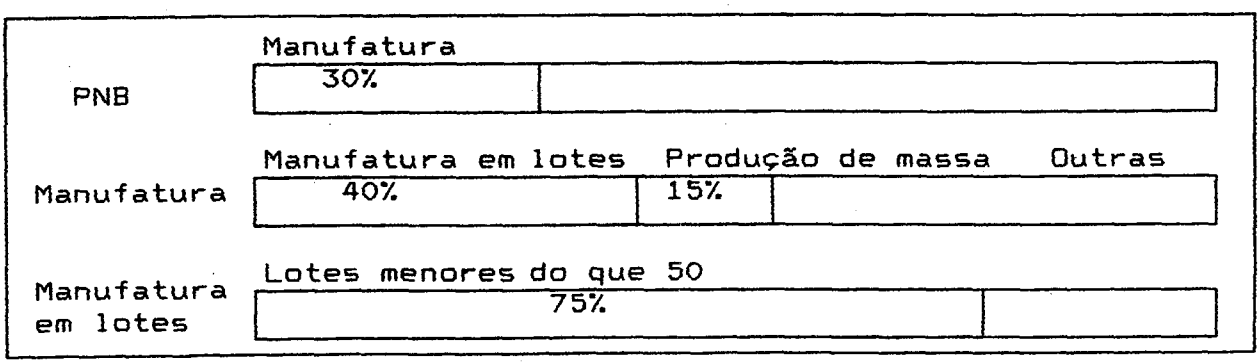
conjunto de peças que pode ser produzido pelo SFM. Então, o analista pode avaliar, através da simulação, diferentes projetos de SFM e selecionar aquele que é melhor para a produção do conjunto escolhido de peças.

Entretanto, no estágio de produção do SFM onde as máquinas ferramentas foram fixadas e o conjunto e peças a ser produzido definido, o objetivo é não mais encontrar a melhor combinação entre peças e máquinas e sim otimizar a utilização de recursos para produzir o conjunto de peças [108]. Neste caso, outros recursos devem ser utilizados.

2.4 - ORIGENS, IMPORTÂNCIA E HISTÓRIA DO SFM

A origem da tecnologia flexível de manufatura reside no desejo de automatizar a produção de pequenos lotes de produtos manufaturados, ou seja, produzir peças em lotes geralmente menores do que 50. Para caracterizar a importância deste tipo de manufatura convém salientar que das nações mais industrializadas, aproximadamente 30% do PNB é representado pela manufatura. Destes, 40% é manufaturada em lotes e somente 15% é de produção em massa. Da manufatura em lote cerca de 75% são de tamanhos inferiores a 50. Tipicamente, portanto, cerca de 10% do PNB das nações mais desenvolvidas está relacionado com a produção de pequenos lotes [90]. A figura 2.8 ilustra as colocações acima.

Figura 2.8 - A importância da manufatura em pequenos lotes.



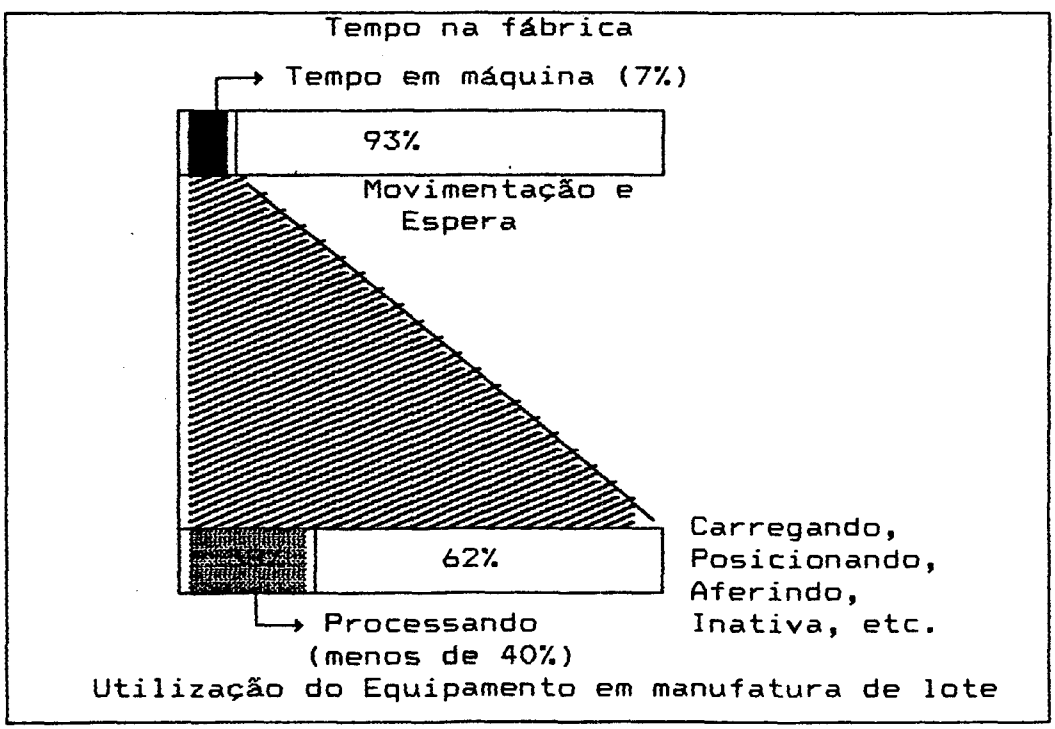
Fonte: GEENWOOD, 1988

Nos Estados Unidos cerca de 50% dos gastos anuais em

manufatura foram realizados na indústria metal/mecânica. Destes produtos, 75% foram manufaturados em lotes de 50 peças ou menos [108]. A produção em lote representa também cerca de 75% da manufatura britânica e 15% do PNB. Além disso, a maioria das companhias manufatureiras britânicas empregam menos do que 500 pessoas e produzem mais do que um produto. Imagem semelhante pode ser feita de outros países europeus [90].

A manufatura tradicional é considerada muito ineficiente para manufaturar pequenos lotes, pois estimativas apontam que de cada peça sendo manufaturada, 93% do seu tempo total de produção é consumido em movimentações e esperas. Dos 7% do tempo que a peça está na máquina, 38% é tempo efetivamente produtivo e os restantes 62% são tempos de *setup* e outros que não de efetivo processamento [91]. Destas estatísticas constata-se que o aumento da produtividade dos volumes pequenos e médios de produção depende do desenvolvimento de sistemas que possam automaticamente estabelecer rotas de peças através do sistema, reduzir tempos de *setup* e operar sem interferência humana, por um turno ou mais. A figura 2.9 ilustra melhor estas estatísticas.

Figura 2.9 - A abordagem tradicional para pequenos lotes



Fonte: GROOVER 1980

O SFM visa este objetivo, uma vez que a flexibilidade é a chave da viabilidade econômica da manufatura automática de pequenos lotes [105].

As deficiências do sistema de manufatura tradicional podem ser enumeradas como [84]:

1. Baixa utilização do equipamento. Estatísticas coletadas mostram que a razão do tempo de efetivo processamento para o tempo total está na base de 7% a 10%, em média. Desta forma, o retorno do investimento para equipamentos caros é correspondentemente baixo, tornando difícil justificar sua substituição.

2. Longos 'lead times'. O avanço dos produtos sendo manufaturados através de uma série de operações individuais em máquinas separadas significa que o tempo de fabricação do início ao fim de uma peça complexa é alto. A isto pode ser adicionado o atraso organizacional devido ao controle necessário em cada estação e a uma folga para registro dos dados para o próximo estágio e tem-se longos *lead times*. No caso de matérias primas caras, isto significa um considerável desembolso de capital sem retorno.

3. Estoques altos. As falhas inevitavelmente conduzem à acumulação de altos estoques de produtos em processo e a custos indiretos associados de armazenagem e auditoria.

4. Inflexibilidade às necessidades de mercado. O longo *lead time* e investimentos em estoque resultam em uma inabilidade para responder rapidamente às mudanças de mercado tanto em quantidade quanto em estilo, a menos que isto possa ser previsto antecipadamente com acerto, o que é muito difícil de ocorrer.

5. Elementos altamente homens-controlado. Operação de equipamentos convencionais são certamente homens-controlados o que torna difícil a manutenção da eficiência produtiva e aumenta a probabilidade de falhas.

6. A manutenção dos níveis de qualidade. A alta intervenção humana no ciclo total de produção leva a uma flutuação da qualidade associada ao fenômeno da "curva de fadiga" e outros fatores.

7. O controle indireto da capacidade. A combinação do grande número de operações de máquinas individuais e a presença de um elemento de controle humano dão origem à custos indiretos de controle.

A tentativa de eliminar estas deficiências acarretaram mudanças no estado da arte, que iniciaram ao final de 1940, e têm

progressivamente sido adaptadas desde então, culminando nos sistemas flexíveis de manufatura como são concebidos hoje [84].

A evolução do sistema de manufatura tradicional para o SFM de hoje não é surpresa. Nos anos 50, presenciou-se a introdução das máquinas de controle numérico [108]. Estas máquinas foram desenvolvidas, primeiramente, nos Estados Unidos, em 1952, pelo MIT (*Massachussets Institute of Technology*) e quase ao mesmo tempo por Alfred Herbert, na Inglaterra [90].

Nos anos 60 surgiram as máquinas de controle numérico computadorizadas ou máquinas CNC e os sistemas de controle numérico direto ou CND, em que um grupo de máquinas ferramentas CN ou CNC são simultaneamente controladas por um computador central. Nos anos 70, sistemas automáticos de movimentação de materiais foram adicionados aos sistemas CNC/CND. Estes sistemas acompanhados por aparelhos automáticos de carga e descarga e trocadores automáticos de ferramentas, resultaram em sistemas capazes de automaticamente transportarem e produzirem uma variedade de peças. Estes sistemas vieram a ser chamados de "Sistemas Flexíveis de Manufatura".

Em resumo, o desenvolvimento do SFM pode ser dividido nos seguintes estágios [103, 116]: (i) criação, 1967-73, (ii) expansão, 1974-79 e (iii) refinamento, 1980 em diante. O estágio de criação pode ser caracterizado pelo emprego de computadores para CND e gerenciamento do tráfego de informações. No estágio de expansão, controladores programáveis foram introduzidos para integrar e atualizar sistemas computadorizados de controle. Nos dois primeiros estágios, a ênfase era, principalmente, sobre o hardware necessário para criar um SFM, máquinas ferramentas, métodos de movimentação de materiais, ferramentaria e fixadores. No último estágio, a ênfase foi no desenvolvimento de sistemas de controle e refinamento dos pacotes de software.

Num ambiente de manufatura, 75% da produção é feita em lotes de 50 ou menos e a tendência em direção a lotes de tamanhos menores é contínua [91]. Lotes de produção na faixa de 10 a 100 peças por hora excedem a capacidade de equipamentos isolados monofusos e são de difícil justificativa econômica para produção em alto-volume das linhas de transferência. É dentro desta faixa de produção, em um ambiente *job shop*, que o SFM encontra o maior potencial para melhorar a produtividade da manufatura.

O rearranjamento físico do *layout* do ambiente produtivo e o uso de sistemas automáticos de movimentação de materiais, combinado com as capacidades programáveis do software de controle do SFM trazem as operações fluxo semelhantes para a orientação discreta do ambiente *job shop* e proporcionam a fabricantes de pequeno e médio porte o potencial para simplificar a administração da produção, melhorar a qualidade e reduzir custos diretos e indiretos de manufatura.

2.5. DIFUSÃO

A maior difusão dos SFM iniciou-se após a recessão de 74/75 dos países ocidentais. Os primeiros usuários da tecnologia foram os Estados Unidos e o Japão. Um grupo inicial de usuários, com entre 50 a 100 sistemas instalados, incluiria o Japão, EUA, URSS, Inglaterra, Alemanha Ocidental e França. Um segundo grupo, com aproximadamente 10 a 20 sistemas instalados, incluiria a Finlândia, Alemanha Oriental, Itália e Suíça.

Tchijow e Sheinin [212] coletaram informações, entre 87 e 88, de publicações de seu país (Áustria) e internacionais, de publicações ocasionais com descrição de sistemas específicos e estimativas de especialistas obtendo banco de dados que inclui mais de 400 exemplos de SFM. As informações seguintes foram baseadas neste banco de dados.

A maioria dos SFM utilizados no mundo são utilizados na indústria metal/mecânica. Cinqüenta por cento são usados na produção de equipamentos de transporte, principalmente para peças de ferro fundido (blocos de cilindros, diferenciais, caixas de câmbio) e peças de aço (válvulas, virabrequins). O segundo tipo de indústria usuária, do setor de transportes, é a indústria aeroespacial e de alta tecnologia. A indústria de máquinas não elétricas é outro grande usuário, com 30% dos sistemas instalados, que são utilizados para a produção de máquinas ferramentas, máquinas agrícolas e de construção. A terceira categoria de indústria é a eletro-eletrônica com 15% dos sistemas instalados e um último grupo é o da indústria de equipamentos de precisão (aparelhos de controle e óticos).

Entre os 375 sistemas, dos mais de 400 existentes, que puderam ser identificados, 83% foram definidos como SFM para

usinagem, 12% para manufatura, 5% para montagem e menos de 1% para moldagem de metais.

A complexidade técnica de um SFM pode ser descrita pelo número de centros de usinagem ou pelo número total de máquinas ferramentas CN e pelo número de robôs utilizados no sistema. A complexidade, combinada com transporte e comunicação dentro de um SFM, armazenagem e inspeção/controle de qualidade depende do tipo do sistema.

Um SFM, normalmente, inclui um ou mais centros de usinagem multi-funcionais. A configuração típica possui de 2 a 4 centros de usinagem e constitui 56% dos 272 sistemas com centros de usinagem. Outros 28% têm de 5 a 8 centros de usinagem, somente 15 sistemas tem um centro complementar formado por outras máquinas CN. O sistema mais complicado é o da Fiat em Termoli, para montar motores, que inclui 72 estações automáticas para a montagem completa de blocos de motores.

Robôs fazem parte de 67 sistemas e destes, 17 incluem somente um. As empresas Casio e Toshiba do Japão e IBM dos Estados Unidos possuem SFM com 70, 72 e 77 robôs industriais, respectivamente. Todas são de produtos eletrônicos.

Aproximadamente 30%, dos 181 sistemas onde existem informações sobre o sistema de movimentação de materiais, utilizam esteiras ou guindastes convencionais para a conexão entre os postos de trabalho. Os outros 70% tem AGVs ou carros controlados por computador.

Dos 42 casos com registros de armazenagem 36 (82%) usam AS/RS (*Automated Storage/Retrieval System*) só 5 casos tem um almoxarifado controlado por computador.

Sistemas de inspeção (33 casos registrados) utilizam medições manuais ou automáticas em 25 dos casos. Em 8 empresas existe um sistema de manutenção e monitoramento automático.

O preço não excede 50 milhões de dólares e 97% custaram menos do que 20 milhões, 16% entre 2 e 3 milhões, 24% entre 1 e 2 milhões e 15% menos do que 1 milhão. Ou seja, mais do que 50% custaram menos do que 3 milhões de dólares.

Apesar do preço mais alto comparado com equipamentos convencionais, o tempo de retorno (*pay-back*) foi curto, ficando entre 2 e 4 anos em 44% dos casos. Um caso na Tchecoslováquia foi de 7 anos.

A flexibilidade expressa pelo número de produtos produzidos por cada sistema, concentrou-se na maioria dos casos em 30 ou menos. Um terço não produz mais do que 10 produtos, 14% de 11 a 20 e 10% de 21 a 30. Existem casos de alta flexibilidade. O SFM da *British Aerospace* produz 2000 variantes de pequenas peças estruturais de aviões em lotes de 5 a 10 unidades cada. Um outro da Toshiba produz 3000 variações de uma caixa comutadora e outros pequenos componentes com 1 a 20 unidades por lote. Os líderes em flexibilidade são dois sistemas Tchecos que produzem 5000 variantes de peças com tamanho máximo de lote de 30 unidades.

O tamanho dos lotes varia de 1 a 50 em 60% dos casos. Em 10%, dos 89 casos onde esta informação estava disponível, o tamanho varia entre 51 a 100, 7% entre 101 a 299 e 9% entre 200 a 500 unidades.

Mais de 60% dos SFM funcionam 3 turnos por dia, 5 ou 6 dias por semana. Somente 44 casos de operação automática foram registrados. Um sistema foi utilizado por 21 horas por dia sem operador.

2.6. VANTAGENS

A principal vantagem do SFM é a sua flexibilidade, que é definida como a habilidade de competir com sucesso, alterando-se as circunstâncias ou em face das instabilidades causadas pelo ambiente [92]. A flexibilidade é um objetivo a ser alcançado por qualquer sistema de manufatura e uma medida crítica da performance total do sistema.

O problema básico da manufatura pode ser descrito como a demanda incerta. O valor competitivo da flexibilidade da manufatura está em sua habilidade de neutralizar os efeitos da incerteza da demanda [92]. A introdução de flexibilidade em um sistema de manufatura requer um investimento inicial alto. Por esta razão, flexibilidade e custo são considerados objetivos conflitantes. Existem muitas maneiras de se conceituar a flexibilidade de um sistema de manufatura. Uma, é defini-la em termos das seguintes características:

1. Flexibilidade das máquinas. É a habilidade de substituir ferramentas quebradas ou gastas, mudar ferramentas em um magazine de ferramentas e montar ou engastar fixadores necessários, sem

interferências ou tempos de *setup* longos.

2. Flexibilidade do processo. É a habilidade para variar os passos necessários para completar uma tarefa. Isto permite que peças diferentes sejam completadas em um sistema utilizando-se de várias máquinas.

3. Flexibilidade do produto. É a habilidade para alterar o sistema de forma a produzir um novo produto com economia e rapidez.

4. Flexibilidade do roteiro. É a habilidade para variar a seqüência de visitação às máquinas (por exemplo, no caso de acidentes) e para continuar produzindo um dado conjunto de tipos de peças, desde que existam várias rotas viáveis de processamento ou que cada operação possa ser realizada em mais do que uma máquina.

5. Flexibilidade do volume. É a habilidade de operar um SFM de forma proveitosa em diferentes volumes de produção.

6. Flexibilidade de expansão. É a capacidade de expandir o sistema facilmente e modularmente.

7. Flexibilidade do seqüenciamento do processo. É a habilidade de alternar a ordem das várias operações para cada tipo de peça.

8. Flexibilidade de produção. É a habilidade para variar rápida e economicamente a variedade de peças para qualquer produto que um SFM pode produzir. Um SFM não obtém flexibilidade de produção até que todas as demais tenham sido alcançadas.

Uma classificação das várias formas que os pesquisadores têm definido flexibilidade da manufatura e as abordagens que foram utilizadas para medí-las pode ser visto em **Gupta** e **Goyal** [92]. Outras fontes relacionadas com a flexibilidade do SFM são [18], [29], [30], [96], [146] e [236].

Flexibilidade pode ser considerada uma vantagem estratégica, cujos resultados serão alcançados com o tempo. Mas, um SFM oferece vantagens imediatas. Algumas delas são a redução:

- dos estoques de produtos acabados e em processo,
- nos tempos de troca,
- de refugos e retrabalhos,
- nos custos de mão de obra,
- do número de amostras e protótipos,
- do consumo de energia,
- do volume de informações a serem processadas,
- do *lead time*,

- das distâncias percorridas pelo produto,
- da área ocupada pela planta,
- dos tempos de *setup*.

Os benefícios acima são aqueles classificados como tangíveis, ou seja, aqueles que se pode fazer uma medição objetiva dos resultados obtidos. Para um exemplo prático dos benefícios obtidos, consultar **Carvalho** [43], onde é relatada instalação de uma célula de manufatura na indústria Metal Leve. Os SFM proporcionam outros benefícios, que não podem, pelo menos à princípio, serem medidos objetivamente. Estes benefícios são denominados de intangíveis. Os principais são [43]:

- utilização de processos padronizados,
- operação mais consistente e menos cansativa,
- introdução mais rápida de novos produtos,
- cumprimento mais eficiente dos prazos de entrega,
- melhoria da qualidade.

Os benefícios acima são relacionados, com pequenas variações por **Green** [89], que acrescenta ainda o aumento da utilização das máquinas e aumento da produtividade. Uma abordagem das vantagens da utilização dos sistemas flexíveis de manufatura a partir do estudo da implantação em um fornecedor do departamento de defesa americano é encontrada em **Fry** e **Smith** [79]. Um relato diferente sobre as vantagens do SFM contrastando o ponto de vista dos tecnologistas com o dos cientistas sociais é encontrado em **Sonntag** [204].

2.7. ELEMENTOS COMPONENTES DO SFM

Um sistema flexível de manufatura é constituído dos subsistemas: processamento, transporte, armazenagem e controle. A separação é um tanto artifical e mais com o objetivo de estudar o sistema, uma vez que muitos equipamentos podem ser enquadrados em mais de um subsistema [39].

2.7.1. Subsistema de processamento

O subsistema de processamento é constituído dos seguintes componentes:

Estação de trabalho. É o local onde é realizada a transformação

física dos elementos e varia de acordo com o tipo de componente sendo produzida. Em sistemas metal/mecânicos, as máquinas são, normalmente, CNC - centros de usinagem horizontal, se peças prismáticas estão sendo produzidas ou centros de torneamento para peças de revolução. Alguns sistemas incluem os dois tipos de máquinas, quando as peças envolvem os dois tipos de operações. Outros sistemas incluem máquinas de propósito único, em contraste com os centros de usinagem, que são capazes de executar várias operações. Além das máquinas de trabalhar metais também são incluídas máquinas para banhos, limpeza, máquinas de aferição e outros tipos para inspeção [39].

Estações de carga e descarga (C/D). Os componentes são introduzidos no sistema em algum ponto. Estes pontos são as estações de carga e descarga onde os componentes são colocadas em *pallets*, normalmente por operadores humanos. Em alguns casos os componentes podem ser fornecidos por um aparelho orientador e carregadas por um robô. A descarga é, normalmente, feita na mesma estação, mas podem existir estações separadas. Em alguns sistemas, estações C/D são dedicadas a um ou mais tipos de componentes, enquanto em outros, elas podem ser utilizadas por qualquer tipo de componente.

Pallets. São equipamentos auxiliares destinados a facilitar o manuseio e transporte dos componentes, bem como, o posicionamento nas máquinas. Dois tipos são comuns. Um, que serve como transportador de pequenos lotes, de modo a facilitar e reduzir a freqüência de movimentos. Ele serve como bandeja, na qual os componentes podem ser colocados nas estações de carga e a partir da qual eles podem ser retirados e colocados na máquina, talvez por um robô. Este tipo é usual em sistemas que utilizam esteiras e pontes rolantes, mas são também utilizados com AGVs. O outro tipo de *pallet* é aquele em que uma ou mais peças são cuidadosamente posicionadas e que é automaticamente movido para a máquina e se mantém na posição enquanto a peça é processada.

Fixadores. São mecanismos utilizados para o posicionamento preciso dos componentes nos *pallets* e/ou nas máquinas. Eles são, geralmente, específicos para um tipo de componente, de modo que cada componente requer um fixador diferente. Em alguns casos, vários tipos de componentes podem ser semelhantes o suficiente para se valerem do mesmo fixador. Fixadores podem estar

permanentemente aderidos aos *pallets* ou podem ser removidos, à medida que um componente requeira um fixador diferente. Se os componentes são pequenos, vários de um mesmo tipo ou de tipos diferentes podem ser fixados num *pallet.*

Ferramentas. São equipamentos que, conectados às máquinas, permitem a transformação física dos componentes. Muitas operações necessitam de ferramentas específicas, tipicamente ferramentas de corte em centros de usinagem. Estes centros possuem magazines de ferramentas em que um conjunto de ferramentas é mantido de modo a executar qualquer tipo de operação de um leque de peças. Ferramentas devem ser trocadas, ou porque acabou sua vida útil ou porque a peça sendo trabalhada requer uma ferramenta que não está no magazine.

Operadores. Apesar dos avanços em direção à fábrica automatizada, muitos, senão todos os SFM necessitam de operadores para várias tarefas, embora alguns sistemas operem automaticamente por um turno, parte de um turno ou mesmo por um dia. Operadores são necessários para carregar e descarregar peças, preparar ferramentas, consertar falhas, tais como, quebra de ferramentas ou má localização de uma peça e ainda para monitorar o sistema.

2.7.2. Subsistema de transporte

Os componentes devem ser transportados de seus locais de carga e descarga para o equipamento produtivo. Os tipos de equipamentos comumente utilizados são as esteiras, veículos e robôs. Nos sistemas de esteiras existem, normalmente, linhas simples ou em *loop*, servindo cada estação de trabalho e algum sistema de endereçamento é necessário para orientar os componentes para as estações de trabalho corretas. Mais populares atualmente do que as esteiras são os veículos, que podem ser de muitos tipos. Vagonetes que ligam as estações através de trilhos, normalmente seguindo caminhos lineares, são encontrados em SFM de pequeno porte. Veículos dirigidos automaticamente, os AGV, orientados por sinais emitidos por fios embutidos no solo ou de estações de comando. Estes veículos toleram *layouts* de caminhos mais complexos. Uma descrição detalhada do funcionamento, aplicações e viabilidade econômica destes meios de transporte pode ser encontrado em **Ferreira e Paskulin** [75]. Alguns podem transportar

uma única carga a cada vez, enquanto outros têm duas posições de carga e os mais especializados podem manejar várias cargas simultaneamente. Existem muitos tipos de robôs móveis com base no solo, que são utilizados para transporte. No entanto, robôs guindastes suspensos são populares, tanto para manejar peças quanto ferramentas [90].

Transporte de ferramentas. A troca e o transporte de ferramentas pode ser feito manualmente, mas sistemas modernos já incluem sistemas automatizados de transporte e troca de ferramentas, que é geralmente efetuada por robôs.

Robôs. São equipamentos articulados e programáveis que são utilizados, principalmente, para o manuseio de peças e ferramentas. Podem ser estacionários ou móveis. Existem vários tipos de robôs móveis mas os dois tipos principais são os fixados sobre trilhos no solo e os em guindastes suspensos. Em alguns sistemas, os robôs são montados em AGVs, que transportam *pallets* de ferramentas para as máquinas, manejando as ferramentas para e do magazine de ferramentas. Robôs também podem ser utilizados para montagem, soldagem ou ainda operações leves de usinagem. As garras dos robôs podem precisar ser trocadas entre o manuseio de uma para outra peça, reduzindo a flexibilidade do robô e do sistema como um todo, impondo operações de lote [90].

2.7.3. Subsistema de armazenagem

A matéria-prima, produtos em processo, ferramentas, *pallets* e fixadores devem ser armazenados no SFM. Esta armazenagem por ser distribuída nas estações ou ao longo do sistema ou então centralizada.

A armazenagem nas estações é realizada pelo *buffer*. Este mecanismo permite que peças sejam levadas para uma máquina que está ocupada e que a máquina continue trabalhando enquanto uma peça processada aguarda transporte. Existem vários tipos de *buffers*. Alguns tomam a forma de dois *stand pallets* em frente à máquina como forma de fornecer uma fila de peças esperando pela máquina e outra de peças esperando transporte para outro local ou máquina. Outros tipos possuem um *pallet* rotatório tipo *shutle* com uma posição para espera e outra para trabalho. Outros, ainda, fornecem um trocador automático ou cadeia de *pallets*, fornecendo

seis ou oito posições de espera.

Além do *buffer* de armazenagem nas estações, muitos sistemas têm estruturas de armazenagem para produtos em processo, *pallets*, fixadores ou matéria-prima, distribuídos ao longo do sistema para facilitar qualquer oscilação no fluxo de trabalho. Em muitos sistemas existem locais de armazenamento para os *pallets*, podendo existir uma posição para cada *pallet* ou o suficiente para abrigar apenas aqueles que não estão em uso.

Uma armazenagem central também pode ser usada para manter produtos pré-preparados antes de um turno não operado e, principalmente, para matéria prima, através de um sistema automatizado AS/RS (*Automated Storage/Retrieval System*) que pode servir para a fábrica como um todo e não só para o SFM.

2.7.4. Subsistema de controle

Controlar o SFM consiste em garantir que o sistema funcionará como o planejado. Em sistemas tradicionais este controle é normalmente feito por operadores humanos. No SFM o controle é principalmente executado por computadores ou equipamentos relacionados como os controladores lógicos programáveis (CLP). O subsistema de controle envolve o processamento (controle) distribuído e em tempo-real do SFM.

Processamento distribuído

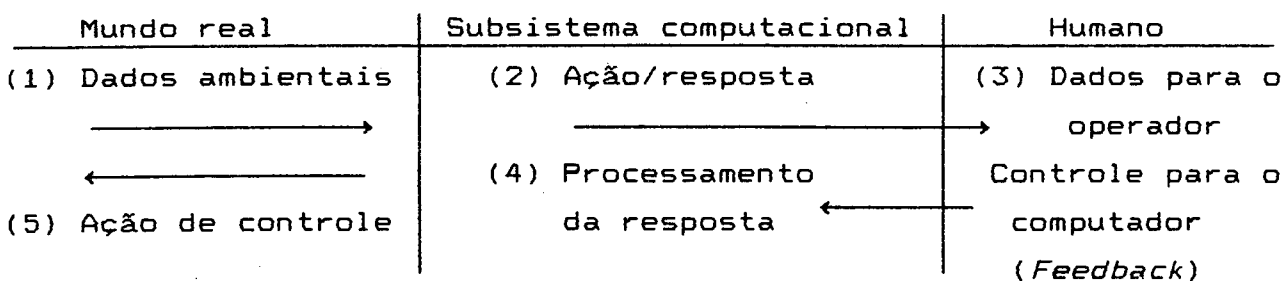
Um processamento é distribuído quando apresentar:
Dispersão. Os recursos de processamento estão fisicamente separados, ou seja, existe uma distribuição física de recursos;
Interconexão: Os recursos de processamento devem ser ligados via uma rede de comunicações e toda comunicação é via a passagem de mensagens na rede;
Compartilhamento. Os recursos de processamento são utilizados por mais de um aparelho de controle [77].

Processamento em tempo-real

Processamento em tempo real é definido como a execução de uma computação durante o tempo atual no qual o processo físico relacionado ocorre, de modo que resultados destas computações

possam ser utilizados na orientação de processos físicos. Isto constitui a noção de controle em tempo-real, isto é, a função do computador em utilizar dados alimentados de sensores no ambiente real para efetivamente guiar a operação do processo físico.

Resumindo tem-se:



Fonte: **FORTIER, 1985**

O subsistema de controle de um SFM é composto de:

(1) Um sistema computacional principal (*host computer*), também denominado de executivo, cuja tarefa é coordenar as atividades das várias estações de trabalho;

(2) Uma rede de comunicação, cuja tarefa é ligar o sistema computacional principal com as estações e

(3) Unidades para o controle detalhado dentro das estações, como unidades CNC para máquinas ferramentas ou robôs ou ainda controladores lógicos programáveis (CLP).

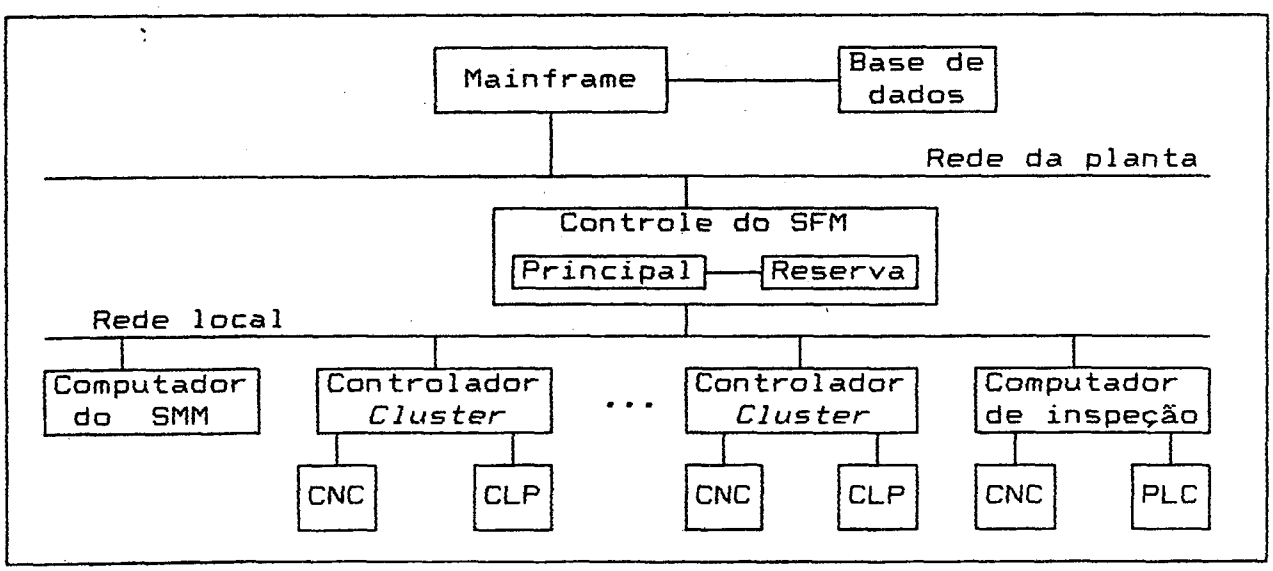
Normalmente, é necessário ter um mecanismo adicional para interfacear os aparelhos de controle das máquinas ferramentas e demais estruturas com a rede de comunicações do sistema. Estes aparelhos são denominados **unidades de interface** e são necessários porque muitas unidades de controle não são capazes de suportar comunicação real com o SFM, isto é, permitir a transmissão e recepção de programas de peças, informações do estatus e instruções de controle.

As tarefas do computador principal ou executivo são variadas. A função básica é dar instruções para máquinas e transportadores. Monitorar o estatus do sistema de modo que isto possa ser feito é uma função vital. Muitos computadores executivos fornecem instruções aos operadores na preparação de ferramentas e estações de C/D e para tomar decisões de programação. Para poder programar o trabalho, o computador deve ser suprido com as necessidades de produção e dados da disponibilidade de materiais.

Isto pode ser obtido unindo-o com o sistema computacional principal de controle da produção da fábrica. O computador também deve fornecer relatórios adminstrativos do trabalho feito, produtividade do sistema, diagnósticos e assim por diante. CLP podem ser utilizados para controlar pequenos subsistemas ou secções do sistema, podendo existir controladores separados para os AGVs em sistemas com vários veículos.

A rede de comunicação é o ponto crítico dos sistemas automatizados. Existem muitos tipos diferentes de comunicações que precisam ser feitas, dentro do SFM e do SFM para o exterior, de forma a se alcançar o denominado ambiente CIM. Estas comunicações possuem características diferentes e sua integração tem sido e é o principal entrave para o avanço dos ambientes automatizados. A situação tem evoluído com os esforços da *General Motors* para tornar sua rede de comunicação MAP um padrão. O NBS (*National Bureau of Standards*) dos EUA tem executado um extensivo projeto de pesquisa nas arquiteturas dos sistemas de controle para o SFM e tipos semelhantes de sistemas integrados [211]. As comunicações entre os vários 'centros de inteligência', dentro e fora do SFM, tendem a ser bidirecionais. Estes centros são operadores, CLP, interfaces, computador executivo, micros, CAD/CAM, MRP, etc. Uma arquitetura típica para um SFM é mostrada na figura 2.10.

Figura 2.10 - Arquitetura de comunicação no SFM

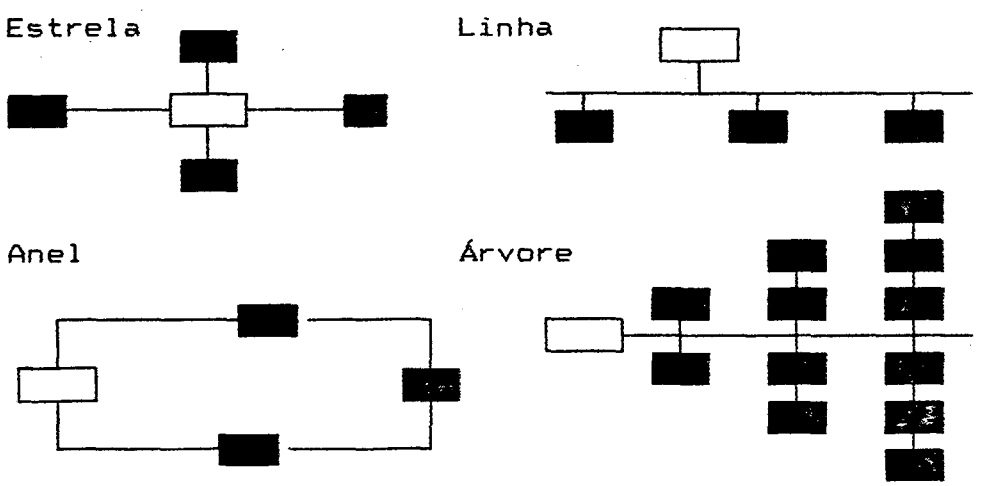


Fonte: SAUL, 1985

A arquitetura do hardware/software do sistema de controle do SFM, ditará a flexibilidade que pode ser obtida. Três possibilidades podem ser consideradas, partindo de um sistema centralizado para um totalmente distribuído [186].

- No sistema centralizado, o sistema de controle está totalmente implementado em um único computador, provavelmente um mini. Esta máquina gerencia a base de dados e todas as funções de controle. Além disso, toda a aplicação e protocolo de tarefas requeridas pelas máquinas do sistema são executadas neste computador.
- No sistema de rede local, LAN (*Local Area Network*) o controlador será um mini ou um ou mais micros, e o controle é distribuído através de microprocessadores. A secção de protocolo do software da máquina dependente está contido na unidade de interface da rede, NIU (*Network Interface Unit*), que é ligada à rede. O controlador do sistema é responsável pela base de dados e pelo software de controle das máquinas.
- No sistema inteiramente distribuído, a base de dados e as NIU são conectadas diretamente à LAN. A aplicação e o protocolo de software que fornece a inteligência para operar a máquina é colocada na NIU. O controle do sistema tem somente a função de controle do SFM. A LAN é a ligação entre o software de controle do sistema e a máquina real sendo operada. A responsabilidade dos programas de peças é passado do processador do sistema de controle para os NIU. Os principais tipos de arquitetura de rede podem ser vistas na figura 2.11.

Figura 2.11 - Tipos de redes



Fonte: GREENWOOD, 1988

Os quatro tipos mais comuns de arquitetura de comunicação dentro de um SFM são, estrela, linha (*bus*), anel e árvore [90]. A arquitetura de árvore é a mais nova adição. Derivada da estrutura em linha, é utilizada no ambiente MAP. Uma comparação entre os tipos de rede é mostrada no quadro 2.1.

Quadro 2.1 - Comparação entre as redes de comunicação

Tipo	Instalação			Interfaces	
	Comprimento	Cabos	Custos	Velocidade	Custo
Estrela	Longo	Muitos	Alto	Baixa	Baixo
Anel	Curto	Poucos	Médio	Rápida	Médio
Linha	Curto	Poucos	Baixo	Média	Alto
Árvore	Médio	Médios	Alto	Alta	Alto

Fonte: GREENWOOD, 1988

2.8. ILUSTRAÇÃO

Um exemplo de SFM é o sistema **Tugefa** da *Brown Bovery & Cia* em Baden, Suíça, onde são manufaturadas estruturas médias e grandes de turbopropulsores. Ele consiste de cinco centros de usinagem, duas estações de carga e recarga, uma estação de limpeza, uma estação de rebarbamento, uma estação de descarga e um *buffer* para armazenagem intermediária de peças. Existem ainda dois transportadores controlados automaticamente (AGVs) que fazem a movimentação de materiais em processo (peças e ferramentas). A supervisão global do sistema é realizada por um minicomputador PDP 11/44. No topo da hierarquia computacional está um mainframe IBM que também é utilizado para aplicações comerciais e administrativas . O IBM é conectado a um mini (VAX 730) onde todos os programas NC estão armazenados. O VAX é conectado ao computador do SFM que por sua vez é conectado a um PDP 11/23 resposável pelo gerenciamente dos AGVs [16].

CAPÍTULO III

3. A PROBLEMÁTICA DOS SISTEMAS FLEXÍVEIS DE MANUFATURA

3.1. INTRODUÇÃO

O ciclo de vida do SFM pode ser caracterizado pelos problemas que devem ser resolvidos ao longo de sua implantação e operacionalização. Estes problemas são agrupados em quatro classes que variam dos de longo prazo para os de prazo muito curto. O primeiro tipo é constituído pelos problemas de longo prazo, que são os de projeto, o segundo tipo é constituído pelos problemas de médio prazo e representados pelo planejamento, o terceiro tipo são os problemas de curto prazo e representados pela programação e, por fim, os problemas de prazo muito curto (em tempo real) representados pelo controle do SFM [121, 197, 202].

Estes problemas são resolvidos principalmente por simulação, mas também por simulação combinada com modelos analíticos, notadamente programação linear [173] e inteira [204] e redes de filas [59]. Com relação a simulação existe uma tendência em direção a utilização de múltiplos modelos [49]. O objetivo é ajustar o esforço em utilizar o modelo aos níveis de precisão e detalhes exigidos pelos resultados. Modelos **macros**, em um nível agregado de detalhes, são utilizados nos primeiros estágios do projeto como, por exemplo, em [33, 40, 41, 75], para determinar a viabilidade global, dar respostas aproximadas e fornecer idéias iniciais das cargas das máquinas, gargalos e adequação do SMM. Para responder questões específicas como o 'número de veículos de movimentação de materiais' ou 'os melhores algoritmos de atribuição', são utilizados modelos ao nível **micro** como, por exemplo, em [85, 228]. A problemática envolvendo o estudo dos sistemas flexíveis de manufatura, juntamente com exemplos de estudos de simulação encaminhando estes problemas, é colocada a seguir.

3.2. PROBLEMAS DE PROJETO

A utilidade da simulação no projeto de sistemas de

produção (SFM) tem sido reconhecida por muitos fabricantes. Vendedores de software notando esta demanda estão começando a unir softwares de CAD/CAM com ferramentas de simulação de modo que os projetistas possam avaliar de uma forma mais consistente o efeito de mudanças nos projetos dos sistemas de produção [26]. Como problemas de projeto de um SFM podem ser incluídos [112, 202]:

3.2.1. O problema do layout

Dentro dos problemas de projeto o principal está relacionado à definição do *layout*. O problema do *layout* está associado aos seguintes sub-problemas:

- Projeção de custos para avaliar várias mudanças na configuração do sistema e
- Pré-teste operacional de configurações alternativas para decisões em: aquisições de novas plantas, substituição, adição, abandono e modernização de equipamentos.

Como visto no capítulo anterior, um grupo de máquinas ou o *layout* de uma planta pode ser arranjado em uma 'posição fixa', 'processo', 'fluxo do produto', ou ainda, 'celular'. Um *layout* pode ser, ainda, uma combinação de duas ou mais destas estratégias.

À parte da questão da obediência à normas de segurança e a princípios gerais de planos de trabalho, um *layout* eficiente das instalações deve satisfazer, alguns ou todos, os seguintes critérios [71]:

a) Custo mínimo de manuseio. Pode ser obtido pela escolha adequada do sistema de movimentação de materiais (SMM) e a localização relativa dos centros de trabalho, departamentos, estações de carga e descarga e pontos de armazenagem do sistema. Os custos variam de acordo com dois fatores principais: o número de unidades de carga transportadas por unidade de tempo e a distância percorrida pelo material.

b) Maximização dos movimentos seqüênciais. O trabalho deve fluir somente numa direção, deve-se eliminar sempre que possível os movimentos de retorno.

c) Minimizar os custos de capital. Minimizar o capital investido em equipamentos e espaço, não só o de chão, mas também o de volume (aéreo).

d) Minimizar custos operacionais. Além dos custos de movimentação

de materiais e do trabalho direto, o total de *setup*, tempos de troca, estoques em processo e paralizações são custos que devem ser minimizados.

e) Máxima flexibilidade. Um *layout* deve ser flexível no sentido de que ele possa ser rapidamente modificado para se adaptar a novas situações.

O projeto inicial do *layout* é um problema complexo com objetivos múltiplos e conflitantes, bem como entradas que podem ser de natureza quantitativa ou qualitativa. Para gerar 'configurações' do *layout*, além do método manual, existem muitos algoritmos diferentes e programas computacionais. Entre os mais populares destes algoritmos/programas computacionais temos: **Craft**, **Corelap**, **Aldep** e **Blockplan** [230].

Estes algoritmos aceitam entradas tais como os fluxos projetados entre os departamentos e fornecem como saídas vários *layouts* candidatos. O Craft, por exemplo, tem por base um problema de atribuição quadrático [122] para a formulação da configuração do *layout*. Pode-se, desta forma, gerar várias configurações de candidatos a *layout* e, então, utilizar simulação para avaliar a 'melhor'. Um exemplo da utilização do Craft integrado com simulação é encontrado no simulador **Fmsds** [230]. **Almodovar** [6] investiga a utilização dos simuladores de manufatura para a avaliação de configurações de *layout*.

Além do problema do *layout*, os seguintes devem ainda ser considerados:

3.2.2. O problema organizacional

O problema organizacional consiste em determinar os tipos de peças a serem manufaturadas no SFM em função das restrições tecnológicas. Os planos de processos para cada tipo de peça devem ser estabelecidos, de modo a determinar a capacidade e as necessidades funcionais do sistema, sugerindo, por sua vez, as várias combinações de máquinas e ferramentas de corte que devem ser utilizados para executar as operações necessárias. Os conceitos de GT podem auxiliar nestas decisões.

É importante, também, padronizar as ferramentas necessárias para manufaturar as peças. Quanto maior a variedade de peças, maior será a variedade de ferramentas e fixadores. Para manter a variedade e o número total de ferramentas e fixadores

dentro de limites aceitáveis, é vital racionalizar, tanto quanto possível, no projeto do componente, métodos de fixação, tamanho de perfurações e assim por diante. Em **Avonts** [11] encontra-se um exemplo da utilização de modelo híbrido de simulação e programação linear para decidir que produtos produzir em um SFM e em que quantidade.

3.2.3. Seleção de um sistema de movimentação de materiais

A seleção de um sistema de movimentação de materiais (SMM) é uma decisão de grande impacto na performance global do sistema, uma vez que, conforme visto, em sistemas tradicionais, a maior parte do tempo de um componente no sistema é gasto com movimentação. Além de influir diretamente na perfomance do sistema, um terço do custo do produto é absorvido pelas várias tarefas de movimentação de materiais (excluindo o custo final de distribuição) [90]. As cargas a serem transportadas dentro do sistema incluem peças, fixadores e moldes, ferramentas, lubrificantes e líquidos refrigerantes, cavacos e refugos, pessoas e equipamentos diversos.

Deve ser decidido o(s) tipo(s) de SMM, tanto em número quanto em capacidade, observando que alguns tipos servem como *buffers* de armazenagem. Os principais tipos de SMM podem ser vistos no quadro 3.1.

Quadro 3.1 - Características dos principais tipos de movimentação de materiais

Caracter. / Tipo	Tipo de carga	Capac. de carga	Veloc.	Flexib. de Rota	Custo	Versatil. Global
AGVS	Disc.	Alta	Média	Alta	Mto Alto	Alta
Vagonetes	Disc.	Alta	Alta	Baixa	Alto	Baixa
Esteiras	Cont.	Md/Baixa	Bx/Alta	Média	Bx/Alto	Muito Alta
Robôs	Disc.	Md/Baixa	Média	Baixa	Md/Alto	Média
Guindaste Rolante	Disc.	Md/Baixa	Média	Baixa	Bx/Alto	Baixo
Manual	Disc.	Baixa	Baixa	Mto alta	Baixa	Muito Alta
Empilha-deiras	Disc.	Md/Baixa	Média	Alta	Baixa	Alta

Fonte: **GREENWOOD, 1988**

O número de estudos realizados para simular o sistema de manuseio de materiais demonstra a sua importância. Alguns exemplos são:

- Utilização da linguagem SLAM para projetar um SMM em um SFM, analisando o impacto de vários fatores de MM sobre critérios de desempenho convencionais. Os parâmetros de decisão investigados foram:
  - Política de despacho,
  - Armazenagem interna,
  - Velocidade,
  - Número de *pallets*,
  - Velocidade das ferramentas,
  - Tempo de C/D para as peças.

  E as medidas de desempenho avaliadas foram:
  - Utilização das máquinas,
  - Taxa de saída (*throughput*),
  - Utilização dos *pallets*,
  - Tempo médio das peças no sistema,
  - Tamanho das filas.

  Quinze diferentes tipos de políticas foram simuladas para estimar o impacto dos parâmetros de projeto sobre o SFM [46].

- Simulação para calcular o número de AGVs necessários em um SFM [149].

- A utilização de um modelo em Siman para avaliar o projeto de sistemas de AGVs para um *layout* particular [127].

- O uso da simulação para avaliar o projeto e a instalação de sistemas de esteiras livres e propelidas [86].

- Simulação para estudar a integração de AGVs com manuseio de materiais não automatizada utilizando um modelo escrito em Siman [95].

- Outros estudos de simulação de SMM são encontrados também em [98], [162] e [209].

3.2.4. Seleção de fixadores e pallets

A duas principais categorias de peças produzidas em um SFM são as torneadas e as prismáticas [211]. As torneadas são aquelas com simetria rotacional sobre um eixo e as prismáticas são

as restantes.

Num SFM uma peça prismática é, normalmente, presa por um fixador que leva em conta a sua forma. O fixador, por sua vez, é montado em um *pallet* que, geralmente, é a mesa de trabalho da máquina.

Os sistemas flexíveis de usinagem podem ser classificados em usinagem de peças prismáticas e usinagem de peças torneadas e os elementos dos dois tipos de sistemas são significativamente diferentes, quando as peças a serem trabalhadas são pequenas ou médias. Em virtude desta diferença *pallets*, fixadores e máquinas devem ser separadas em equipamentos para usinagem de peças prismáticas e torneadas. Para usinagem de peças grandes não é necessário fazer esta distinção, pois a diferença de elementos em tais sistemas não é significativa [211].

Os *pallets* formam a interface entre a peça e as máquinas e é, desta forma, necessário que todas as máquinas em um sistema aceitem o mesmo tipo de *pallet*. Isto não será um problema se todas as máquinas são idênticas, mas no caso de serem diferentes, deve haver adaptações para que se possa utilizar uma interface padrão.

A configuração de *pallet* mais utilizada é aquela que o *pallet* tem a mesma forma da mesa de trabalho do centro de usinagem (máquina) do sistema, havendo, entretanto, exceções. *Pallets* são caros, pois requerem uma grande quantidade de material para assegurar sua rigidez e estabilidade.

A fixação das peças a serem usinadas geralmente envolve o uso de um fixador retangular. Para posicionar peças fundidas ou forjadas de tamanho médio são utilizados fixadores de estrutura semelhante. Um segundo tipo comum de fixador de forma retangular pode ser denominado de fixador suporte, que possui peças fixadas em seus lados e pode acomodar duas ou mais peças por vez.

O uso de fixadores multipeças reduz o ciclo de máquina por fixador, reduzindo o número de fixadores e *pallets* requeridos pelo sistema com considerável economia de custos. A variedade e complexidade dos fixadores têm sido um dos principais problemas na aplicação do SFM. Um usuário, no final das contas, gasta mais em fixadores do que em todo o restante do sistema, em alguns casos. A simulação de alternativas de arranjo de *pallets* e fixadores mostra-se, desta forma, importante para reduzir significativamente os custos totais do sistema. **Abdin** e **Mohamed** [2] utilizam um

modelo de simulação, escrito em Q-Gert, para determinar o número de *pallets* em um SFM.

3.2.5. Seleção do sistema computacional (Software e Hardware)

Qualquer FMS pode ser decomposto em 3 componentes principais [90]:

1. Equipamentos de produção (processos e máquinas).
2. Uma rede de comunicação.
3. Um sistema computacional de controle.

Os dois últimos são, talvez, os mais importantes e também os mais complexos. Esta complexidade tem sido fruto da maioria dos problemas para o avanço e proliferação dos SFM. Progressos significativos têm sido obtidos no hardware, mas o mesmo não tem ocorrido com o software.

Em termos do custo total de um SFM pode-se debitar de 10 a 15% para o hardware e outros 15 a 30% ao software de controle, sendo razoável sugerir que algo como 50 a 75% do risco total envolvido na implementação do SFM é representado pelo sistema computacional de controle, pois é muito difícil se estimar a capacidade do software antes de sua total instalação e teste [90].

Essencialmente, os objetivos do sistema computacional de controle para qualquer sistema flexível de manufatura são [90]:
(1) Facilitar a transmissão de softwares de suporte (como programas de peças) para os equipamentos de processos dentro do sistema;
(2) Coordenar as atividades dos equipamentos;
(3) Facilitar a entrada de dados, controles, operações e monitoramento do SFM como um todo pelos operadores;
(4) Auxiliar, através de diagnósticos, o retorno do sistema para uma condição operacional integral após a ocorrência de uma falha, uma vez que, a tarefa de reinicializar o sistema dificilmente será totalmente automatizada.

3.2.6. Seleção de um sistema de armazenagem

Mesmo os sistemas que utilizam o JIT (*Just-in-Time*), necessitam de pontos de armazenagem para matéria-prima ou produtos em processo. A decisão vai além do fato de escolher entre armazenagem central ou local ou, ainda, ambas, pois o sistema de

armazenagem está intimamente relacionado com o sistema de movimentação de materiais e mesmo com o sistema de *pallets* e fixadores, uma vez que, *pallets*, fixadores, esteiras, AGVs, etc. podem servir como pontos de armazenagem local.

Um AS/RS é essencialmente um depósito (almoxarifado) completamente automatizado e controlado por computador. Peças são transportadas de e para qualquer dos SMM. A armazenagem fica totalmente dependente do sistema de controle do AS/RS. Estes sistemas tendem a custar muito caro, embora estejam disponíveis em uma larga variedade de tamanhos e níveis de sofisticação. Possuem a vantagem de serem independentes e capazes de manusear um grande número de componentes dentro de uma razoavelmente pequena área de solo.

Uma das principais desvantagens destes sistemas é que se eles quebram (o que é muito raro), a operação manual do sistema fica extremamente difícil, não só por problemas de espaço físico para o operador se movimentar no seu interior, mas também para localizar os componentes armazenados, uma vez que o computador normalmente aloca estes componentes com base em algoritmos sofisticados como forma de assegurar que os mais utilizados fiquem armazenados mais próximos da saída/entrada do sistema.

Além do alto custo destes sistemas, um inventário do seu conteúdo pode ser difícil de se realizar visualmente, uma vez que não se saberá, a não ser quem o opera, se o sistema está cheio, quase cheio ou vazio.

Exemplos da utilização da simulação para solucionar problemas de armazenagem incluem:

- Um modelo em Slam para simular algoritmos de controle para AS/RS fornecendo também diretrizes para projetar estratégias de controle para estes sistemas [129].

- Um modelo de simulação para auxiliar no desenvolvimento do projeto de sistemas AS/RS, consistindo de várias ilhas de armazenagem, aparelhos de movimentação de materiais, *loops* de esteiras fechado, estações de trabalho e *buffers* fornecendo interface com a esteira [161].

3.2.7. Seleção de máquinas ferramentas, acessórios e processos

Uma vez que os produtos a serem produzidos tenham sido

definidos, outros aspectos do projeto do sistema estarão definidos. Por exemplo, se o sistema está sendo projetado para produzir componentes rotacionais, tornos e talvez retificadoras cilíndricas, se componentes prismáticos devem ser produzidos, centros de usinagem e furadeiras, devem estar presentes no sistema [211].

Tendo identificado os produtos, o próximo passo é estabelecer o processo mais apropriado para manufaturar estes produtos e sua viabilidade, elaborando uma detalhada especificação, principalmente, de processos recentemente desenvolvidos.

Executando uma pesquisa dos atuais processos produtivos e equipamentos disponíveis e tendo selecionado a opção mais apropriada, o próximo passo é decidir como as peças deverão ser produzidas nos equipamentos e quantas estações de processamento serão necessárias para acomodar o volume de produção previsto. Para isto, deve-se ter estatísticas dos tempos esperados de processamento de cada peça a ser produzida.

Estabelecido o tipo e o número de ítens dos equipamentos principais a serem incluídos no SFM, a próxima etapa é decidir que tipo de equipamentos auxiliares serão necessários para dar suporte às atividades das estações de processo. É claro que, muitos destes equipamentos são de aplicação dependente, mas algumas opções a serem consideradas são:

- NC, CNC, DNC?
- Comunicação MAP compatível ou outro sistema?
- Equipamentos de aferição e controle de qualidade em processo ou pós processo?
- Troca automática de ferramentas, fixadores e *pallets*?
- Remoção automática de cavacos e refugos?
- Compensação do desgaste de ferramentas?

Uma das principais vantagens do SFM sobre uma planta tradicional é a produção de componentes com variações qualitativas muito pequenas. A seleção dos equipamentos adequados de aferição e a abordagem de inspeção da qualidade é importante para que isto seja alcançado. A seleção dos equipamentos de aferição depende da acurácia necessária dos componentes. Existem quatro abordagens, de inspeção, que podem ser consideradas:

. Inspeção manual

. Inspeção automática

. Em-processo

. Pós-processo

Uma outra consideração dentro do processo são as operações de *setup*, em virtude da produção em pequenos lotes, que devem ser tão curtas quanto possível. Devem ser programadas para ocorrer no tempo mais conveniente, quando todos os recursos estiverem prontamente disponíveis.

Um exemplo de simulador para um sistema de qualidade em um SFM, escrito em Fortram IV, é encontrado em **Palaniswami** e **Hassan** [126].

3.2.8. Justificação econômica

Existe um número considerável de técnicas de avaliação econômica de um SFM. Elas são, essencialmente, derivadas das seguintes abordagens básicas. O método do(a) [90]:

Reembolso (*pay back*). É o tempo necessário para recuperar o investimento dos incrementos de lucros produzidos.

Retorno do investimento. É a percentagem do investimento representado pelo lucro gerado em um ano 'normal' de operação.

Valor presente líquido. Leva em conta o fluxo de caixa (*cashflows*) gerado ao longo da vida do projeto.

Taxa interna de retorno. É semelhante ao valor presente líquido, mas é usada para encontrar a taxa de juros que resultará em um valor presente líquido igual a zero.

MAPI. É usado, principalmente, para investimentos de reposição e é, também, baseado na abordagem do valor presente líquido, ajustado para utilização do capital.

Infelizmente, estas abordagens sofrem sérias limitações, uma vez que a maioria dos valores envolvidos não são conhecidos com algum grau de certeza, especialmente nos estágios iniciais do projeto, que é quando a maioria de tais análises é executada. Além disso, o sistema (ou vários sistemas potenciais) é somente hipotético no estágio de projeto e a determinação dos custos e benefícios de tais sistemas não pode ser feita pela observação do desempenho. Como conseqüência a simulação vem sendo, também, utilizada para avaliar economicamente um SFM.

Exemplos de estudos de simulação para a justificação econômica de um SFM incluem:

- Uma metodologia que se vale da teoria das filas e simulação para a justificação econômica de um SFM. A metodologia é um procedimento em duas fases, onde a primeira é um modelo interativo, baseado na teoria das filas para resolver os problemas da seleção das peças e planejamento das necessidades de máquinas. A configuração do sistema gerada na fase um é avaliada num modelo de decisão que constitui a fase dois. A cada ponto de decisão, o decisor interage com o modelo de simulação para especificar cursos de ação alternativos para serem avaliados [57].

- Uma comparação do ponto de vista econômico entre os sistemas convencionais e os flexíveis de manufatura [168].

- Um modelo de otimização estocástico, dinâmico para o processo de decisão de investimento em manufatura automatizada flexível. O modelo é utilizado para analisar as propriedades qualitativas de políticas ótimas para aquisição de automação flexível. Resultados tanto analíticos quanto de simulação são fornecidos para ilustrar as propriedades e validar parcialmente as especificações do modelo [142].

3.3. PROBLEMAS DE PLANEJAMENTO

Resolvidos os problemas de projeto surge um segundo tipo de problema representado pelo planejamento do sistema. Existem decisões que necessitam serem tomadas antes da produção ser iniciada [197, 202].

Iniciar a produção em um SFM envolve, principalmente, a seleção: das máquinas ferramentas, do conjunto de ferramentas disponíveis, das peças a serem produzidas e dos *pallets* e fixadores nos quais elas serão montadas e fixadas. Isto significa que quase todos os componentes do sistema estão envolvidos e implica em que tratar este problema na totalidade é praticamente impossível [201]. A solução é desagregar os problemas de planejamento em subproblemas. O que são de fato problemas de planejamento varia muito de autor para autor, no entanto, pode-se obervar que vários [197, 234] seguem a abordagem dada por Stecke [201]. Para estes autores o problema do planejamento é composto dos seguintes subproblemas.

3.3.1. Seleção do tipo de peça

Consiste em determinar do conjunto dos tipos de peças que deve ser fabricado (ou por demanda prevista ou por pedido dos consumidores), um subconjunto de tipos de peças para manufatura imediata e simultânea.

3.3.2. Agrupamento das máquinas (*pooling*)

Consiste em particionar o conjunto das máquinas ferramentas em grupos de tal forma que cada máquina em um grupo particular possa executar o mesmo conjunto de operações. Os benefícios do agrupamento são, por exemplo, permitir rotas alternativas no caso de impedimento de uma máquina e, também, para que a quebra de uma máquina não paralize a produção.

3.3.3. Mix da produção

Consiste em determinar as proporções relativas na qual os tipos de peças candidatas à produção devem ser produzidas como forma de maximizar as saídas (*throughput*).

As duas principais abordagens propostas para resolver os problemas do mix da produção são, por um lado, a abordagem do lote (*batching*) onde os tipos de peças serão particionados em lotes separados que serão processados seqüencialmente, com cada lote podendo conter vários tipos de peças e, por outro lado, a abordagem flexível, autorizando a introdução de um novo tipo de peça no SFM, quando a produção total de um tipo tiver terminado.

A divisão em lotes limita a variedade de peças fluindo no sistema e isto é desejável por duas razões principais:

- Poucas classes de peças fluindo no sistema significa que poucas ferramentas não idênticas são requeridas, a qualquer tempo, pelo sistema diminuindo os tempos de troca.
- Uma menor variedade de peças significa menos diversidade de rotas e, desta forma, menos incidência de bloqueamento/travamento.

O loteamento altera o fluxo padrão de peças no sistema. No limite, se todos os lotes são homogêneos (um tipo de peça por lote), o SFM é operado como um *job shop* ou como uma linha de transferência programável, simplicando o problema do controle. Um modelo híbrido de simulação, programação inteira e redes de filas para selecionar o mix das peças é relatado em **Stecke e Kim** [204].

3.3.4. Alocação de recursos

Consiste em alocar os limitados recursos (como *pallets*, fixadores e ferramentas) entre os tipos de peças selecionadas. Quando as peças requerem operações com dois ou mais fixadores, os engenheiros de produção devem considerar se os sucessivos estágios estão sincronizados de alguma forma.

Em um ambiente de manufatura automatizado, os equipamentos têm flexibilidade que não existe na maioria das plantas de manufatura tradicional. Um exemplo disto é em usinagem, onde cada centro de máquinas CNC é instalado com um arranjo de ferramentas inserido em um magazine, e cada ferramenta é selecionada para uso de acordo com o programa que controla a máquina. A flexibilidade deriva de configurar o equipamento com as ferramentas e, então, extrair destas ferramentas a máxima utilização sem a necessidade de troca de peça para peça. No entanto, a flexibilidade é limitada (o magazine de ferramentas tem *slots* limitados). A flexibilidade do equipamento de manufatura depende da configuração de seus recursos disponíveis, daí a importância de configurar a capacidade limitada otimamente para obter o máximo da flexibilidade potencial.

O problema da alocação de recursos é um problema puro de atribuição ocorrendo no nível físico do SFM. Ele pode ser facilmente resolvido, quando as restrições sobre a disponibilidade de fixadores e *pallets* são levadas em conta ao resolver os problemas da seleção do tipo de peça e da taxa de produção [49].

3.3.5. Da carga (*loading*)

Consiste em alocar as operações e as ferramentas associadas do conjunto selecionado de peças entre os grupos de máquinas sujeitos às restrições tecnológicas e de capacidade do SFM.

A solução do problema da carga deve sujeitar-se a certas restrições [205]:

- Cada operação requerida e todas as ferramentas associadas devem ser atribuídas para, no mínino, uma máquina;
- Uma operação pode ser atribuída somente para aquelas máquinas capazes de executá-la;
- As ferramentas necessárias para o conjunto total de operações

atribuídas para qualquer máquina não devem exceder a capacidade do magazine de ferramentas desta máquina.

E para melhorar o desempenho do sistema:

- As cargas de trabalho atribuídas para cada máquina devem ser balanceadas (em algum sentido) para evitar estrangulamentos desnecessários;
- Quando viável, operações consecutivas devem ser executadas na mesma máquina para minimizar o número de movimentos de peças requeridas;
- Se o magazine de ferramentas permitir, as operações devem ser atribuídas para mais do que uma máquina para aumentar a flexibilidade quando roteirizar peças em tempo real.

O problema da carga tem sido tratado por simulação por vários autores. Alguns exemplos são:

- Um estudo de simulação para um tipo dedicado de SFM onde são examinandas cinco estratégias de carga versus 16 regras de despacho. O estudo chegou a várias conclusões significativas sobre como o sistema deve ser controlado e indica que a escolha da carga aplicável e das estratégias de programação dependem de muitas variáveis particulares do sistema [205].

- Um modelo de simulação para ilustrar um modelo analítico e para estudar os efeitos da carga no desempenho do sistema sob diferentes regras de despacho [190].

3.4. PROBLEMAS DE PROGRAMAÇÃO

Os problemas subseqüentes do sistema, após ele ter sido preparado durante o estágio de planejamento, estão relacionados com a sua operação.

O problema da programação envolve o seqüenciamento de peças e de cada recurso de produção, sujeito a restrições de procedência, prazos, trabalho em processo, etc. O objetivo típico é minimizar o fluxo médio de tempo, nível de estoque, atrasos, etc. [48].

As diferenças entre programar um *Job Shop* convencional e programar um SFM, são mínimas, com uma exceção. Tecnologia de manufatura flexível reduz o tempo de *setup* a um mínimo. Conseqüentemente, o estoque de trabalho em processo pode ser

tomado como um mínimo. Com baixo nível de trabalho em processo (relativo ao nível de *jobs shops* convencionais), a decisão de seqüenciamento não é tão crítica como em um *job shop* [48].

Os principais problemas de programação que ocorrem num SFM são do(a) [197, 203]:

3.4.1. Seqüenciamento de entrada (*dispatching*)

A ordem de entrada no sistema das peças que foram selecionadas no estágio de planejamento deve ser determinada. As referências [2] e [190], entre outras, tratam deste problema.

3.4.2. Seqüenciamento das operações

Algumas operações dos planos de processos das peças podem ser intercambiáveis. A hipótese usual de uma ordem de precedência fixa é uma limitação na flexibilidade do sistema. Em geral, os componentes produzidos em um SFM necessitam uma seqüência de operações. Em alguns casos, a complexidade das peças requer que operações que poderiam ser executadas em uma única estação de trabalho devam ser divididas entre várias máquinas, devido à capacidade de trabalho envolvida ou às limitações da capacidade das ferramentas nas máquinas. Um aspecto adicional da complexidade dos componentes é o número de ferramentas de corte necessárias em sua manufatura.

3.4.3. Seleção da estação de trabalho

Quando uma operação pode ser executada em várias estações de trabalho, uma delas deve ser determinada.

3.4.4. Seqüenciamento das peças

A peça que deve ser primeiramente processada em uma estação de trabalho deve ser selecionada da fila de entrada da estação.

3.4.5. Seleção do veículo de movimentação de materiais

No caso de se utilizar um AGV o carro que deve apanhar a peça na estação de trabalho deve ser determinado.

3.4.6. Seleção do rota (*routing*)

No caso de vários caminhos alternativos, deve-se determinar um. Definir o(s) plano(s) ou rotas do processo para cada tipo de peça. A referência [16] ilustra este problema.

3.4.7. Seleção do operador

Se uma operação não é totalmente automática, um operador deve ser atribuído a ela.

3.4.8. Determinação do lote (*batching*)

Consiste em determinar o tamanho de cada lote de peças a serem processadas e a composição destes lotes, ou seja, o *mix* de peças que compõe cada lote. Este problema é estudado em [16].

Alguns exemplos de modelos de simulação para estudar problemas de programação do SFM incluem:

- Um modelo misto de simulação e programação linear para estudar os problemas de lotear (*batching*) e roteirizar (*routing*), composto de três módulos principais: o módulo lote que define a lista de peças que serão produzidas no próximo período de tempo. O módulo roteiro que define o(s) plano(s) de processo para cada tipo de peça, minimizando o tempo necessário para produzí-la e um módulo de simulação, que executa a programação (baseado no conjunto atual de regras de decisão). Os problemas de lote e rota são formulados como modelos de programação linear. A integração do módulo de simulação no modelo conceitual permite, por um lado, uma avaliação imediata das decisões e, por outro lado, uma redefinição fácil das restrições do sistema, tais como, ferramentas prontas para uso, disponibilidade de máquinas e configuração do magazine de ferramentas. Estes *feedbacks* informando sobre o estado do sistema são importantes para aumentar a acurácia dos resultados do loteamento e roteamento [16].

- Um modelo de simulação em GPPS/H para investigar a conveniência do desbalanceamento da carga de trabalho para um SFM com grupos de máquinas de tamanhos desiguais. O estudo mostrou que a utilização global do sistema é melhor quando desbalanceado [204].

- Um modelo baseado em uma combinação de técnicas de simulação e

IA para tratar os problemas de planejamento, programação e controle do SFM de uma forma integrada [197].

3.5. PROBLEMAS DE CONTROLE

Por problemas de controle entende-se o contínuo monitoramento do sistema para estar certo de que ele está se comportando como o planejado [202]. Consiste principalmente em criar estratégias para lidar com eventos imprevistos, tais como, quebra de máquinas. O planejamento de operações preventivas de manutenção é também, normalmente, considerado como um problema de controle.

Manter a confiabilidade do sistema é um problema. Pontos de inspeção (checagem da qualidade das peças) e a freqüência da inspeção de peças em processo e prontas devem ser determinados. Procedimentos de monitoração para os processos, bem como os métodos de coleta de dados de vários tipos devem ser especificados. Estimativas de durabilidade de ferramentas e outros equipamentos devem ser revisados e atualizados freqüentemente. Razões para erros em processos devem ser encontradas (isto é, *pallets* e fixadores desalinhados, ferramentas gastas, problemas com rejeitos, etc.).

Os complexos problemas de controle em um SFM surgem da multiplicidade de variáveis a serem monitoradas e da necessidade de informações em operações de tempo real. Os problemas relacionados às informações no sistema são, especialmente importantes, pois um estudo da GM de 1981 constatou que 50% dos custos de novos projetos de automação estão diretamente relacionados com mecanismos de comunicação.

Estudos de simulação com o objetivo de resolver problemas de comunicações e controle do SFM incluem:

- Modelagem das formas de integrar computadores, redes de comunicação e CN, incluindo três das mais importantes, que são: a hierárquica de controle de fila-única ou DNC, hierárquica de controle multi-nível com comunicação ponto a ponto ou CNC e a hieráquica de controle multi-nível com LAN. Estas situações foram estudadas através de um modelo de programação linear inteiro de variáveis 0-1, resolvido pela técnica do *branch-and-bound* para encontrar uma configuração computacional viável de custo mínimo.

Para avaliar o desempenho inicial de tal sistema, a configuração é modelada como uma rede de filas, utilizando o CAN-Q e, então, um modelo de simulação detalhado, com hipóteses mais realistas é realizado através da linguagem Siman, fornecendo estimativas mais acuradas do desempenho do sistema [82].

- O modelamento e análise da performance do sistema de comunicação, através de modelagem híbrida conjugada com 3 ferramentas de análise, sendo dois modelos analíticos, um de programação matemática, e um de rede de filas e um modelo de simulação em Siman [83].

- A simulação como uma ferramenta para desenvolver software de controle para o SFM. A lógica é testada e desenvolvida no modelo de simulação. Para testar a comunicação entre as diferentes máquinas e controladores dos transportadores, desenvolve-se emuladores que agem exatamente como os controladores das máquinas e são capazes de se comunicar com o software de controle. A simulação gera dados usados para avaliar a qualidade da estratégia e, neste caso, o mesmo software é usado para simulação e controle. Isto significa que o programa é capaz de comunicar-se com máquinas e controladores de transporte [48].

- Uma combinação de simulação com um modelo analítico para auxiliar no planejamento da produção e programação em tempo real. Os modelos podem ser utilizados independentemente para analisar o comportamento do SFM, mas o principal objetivo é utilizá-los de forma combinada em um modelo híbrido [32].

3.6. CONSIDERAÇÕES

Um levantamento, limitado aos Estados Unidos, da utilização da simulação para estudar os problemas do SFM, constatou que os principais vendedores de SFM consideram simulação como parte essencial dos seus projetos e que não é usual construir um SFM sem estudos prévios de simulação para determinar o impacto dos parâmetros de projeto na dinâmica do sistema. Não somente a simulação é utilizada por vendedores para projetar o sistema mas sua aceitação é tal que muitos manufatores usando ou planejando usar SFM, consideram a simulação como parte integrante de suas operações de planejamento e controle [105].

CAPÍTULO IV

4. TÉCNICAS DE PO UTILIZADAS PARA ANALISAR UM SFM

4.1. INTRODUÇÃO

Os sistemas flexíveis resultaram do desejo de automatizar a manufatura de pequenos lotes com o objetivo de alcançar a eficiência da produção de massa sem perder a flexibilidade da produção *job shop* para processar simultaneamente vários tipos de peças.

Um SFM é um sistema automático de manufatura de lote, consistindo, normalmente, de máquinas NC, ligadas por equipamentos automatizados de movimentação de materiais, que executam as operações necessárias para manufaturar peças. Cada operação requer uma ou mais ferramentas de corte. As ferramentas para quase todas as operações que podem ser executadas por uma máquina são armazenadas em seus magazines de ferramentas. Cada máquina tem um mecanismo automático de troca de ferramentas que pode trocar duas ferramentas de corte em segundos. Esta capacidade de troca rápida permite que várias operações consecutivas sejam executadas virtualmente sem tempos de *setup* entre operações. Um ou mais computadores controlam muitas atividades, tais como as operações de máquinas, movimentos das peças e troca de ferramentas. O computador não pode encaminhar uma determinada peça para uma máquina sem que todas as ferramentas necessárias para a próxima operação tenham sido colocadas no magazine. Esta última restrição mostra a necessidade de planejamento prévio da produção.

Em virtude dos conceitos e tecnologias da manufatura automática de lote estarem ainda em fase inicial, muitos problemas têm sido enfrentados. A necessidade de um planejamento minucioso foi percebido desde a instalação do primeiro SFM [201].

Administrar a produção em um SFM é mais difícil do que em linhas de produção e *'job shops* porque:

- Cada máquina é completamente versátil (flexível) e capaz de executar muitas operações diferentes;
- Cada sistema pode processar vários tipos de peças simultaneamente e

- Cada peça pode ter rotas alternativas através do sistema.

Estas capacidades adicionais e opções de planejamento aumentam tanto o número de variáveis decisórias quanto as restrições associadas com a preparação do SFM.

Para melhor utilizar as capacidades destes sistemas, uma cuidadosa preparação é necessária para iniciar a produção. A organização da manufatura de lote é freqüentemente modificada para satisfazer novas necessidades de produção. Isto constrasta com o sistema de produção de massa onde a organização é parte do projeto e mudanças de preparação são poucas.

As variáveis decisórias do problema de preparação de um SFM são: próximos tipos de peças a serem produzidos, número relativo de peças de cada tipo que serão processadas simultaneamente, número de pallets e fixadores de cada tipo que devem ser reservados para cada tipo de peça e alocação de ferramentas e operações para as máquinas. O objetivo da preparação depende do sistema mas, normalmente, é maximizar a produção esperada.

Os sistemas iniciais foram administrados por meio de métodos convencionais de programação de carga, tais como, atribuir cada operação a somente uma máquina e procurando balancear a carga de trabalho atribuído a cada máquina. Isto dava a impressão que, talvez, novas estratégias de carga e controle poderiam ser desenvolvidas para tirar proveito da capacidade das máquinas e da flexibilidade do sistema.

Como resultado, inúmeros modelos dos mais variados tipos têm sido formulados para estudar os SFM. Modelos são úteis para identificar fatores chaves que afetarão o desempenho do sistema e para se entender como o mesmo se comporta e como os seus componentes interagem. Modelos são aplicados como auxílio para determinar os procedimentos apropriados para projetar e instalar um sistema ou estratégias para auxiliar a utilização do sistema de forma eficiente.

O propósito deste capítulo é examinar os tipos de modelos que têm sido utilizados para analisar os problemas do SFM, relacionados no capítulo anterior e, normalmente, utilizados combinados com simulação. Cada modelo aborda os problemas de um ponto de vista diferente. Alguns modelos agregam vários problemas, enquanto outros os subdividem. Outros, focalizam situações

particulares, normalmente, valendo-se de hipóteses simplificadoras. Outro objetivo do capítulo é situar a codificação de um modelo de simulação no contexto de um estudo completo de simulação.

Um exame da literatura sobre os sistemas flexíveis de manufatura permite identificar as seguintes metodologias de Pesquisa Operacional, utilizadas isoladamente ou, geralmente, acompanhando um estudo de simulação para abordar os problemas relacionados à sua implantação, operacionalização e controle:

- O método heurístico,
- Programação matemática,
- Redes de Petry,
- Inteligência Artifical (Sistemas Especialistas),
- Análise de sistemas dinâmicos a eventos discretos.
  1. Abordagem analítica;
     a. Análise de redes de filas,
     b. Análise do valor médio,
     c. Análise de perturbação,
  2. Abordagem de simulação;
     a. Modelagem física,
     b. **Simulação digital e gráfica.**

4.2. MÉTODO HEURÍSTICO

A primeira metodologia utilizada para analisar um SFM foi o método heurístico. Este método depende largamente da criatividade, experiência e conhecimentos do modelador ou projetista. Enquanto a tecnologia do SFM estava em sua infância, muitos projetistas e usuários se valiam do conhecimento empírico ou heurístico. A maioria da literatura inicial em SFM relata histórias de sucessos por vendedores e usuários e avaliações qualitativas das vantagens e enganos da nova tecnologia [186]. Sistemas flexíveis de manufatura são complexos e regras práticas que funcionam bem com *job shops* e linhas de transferência convencionais podem não funcionar com estes sistemas. Além disso, as necessidades de dois sistemas semelhantes não são as mesmas, desta forma, confiar que um sistema dará certo porque o similar foi bem sucedido é um risco que pode sair caro. Contudo, sucessos

anteriores e histórias de desastres fornecem um primeiro passo para o entendimento do funcionamento dos sistemas flexíveis de manufatura. O método heurístico pode ser incorporado a softwares para construção de *layouts*, combinado com um modelo de simulação para avaliar estes *layouts*.

4.3. PROGRAMAÇÃO MATEMÁTICA

A abordagem da programação matemática para o problema do projeto/implementação do SFM é, geralmente, restrita pelo grau de interação entre os componentes do sistema e as numerosas questões das estruturas do projeto/operação. Algumas destas questões entrelaçadas incluem: decisões sobre componentes do *layout*, considerações sobre movimentações de materiais e unidades de carga, estoque em-processo, carga/programação de máquinas, tamanho de lotes, métodos de trabalho e alocação de recursos.

Estas interações multi-face, geralmente, tendem a tornar abordagens analíticas intratáveis. Por exemplo, o problema de planejamento, de programar um conjunto de peças, ferramentas e máquinas de modo a otimizar algumas medidas de desempenho, pode conduzir a um problema NP-completo [108]. Como conseqüência, questões de planejamento e implementação são tipicamente resolvidas iterativamente. A natureza do processo de produção, por exemplo, afeta a formulação e análise do problema de planejamento das necessidades de máquinas. Se o plano do processo é especificado, então, alguma característica operacional juntamente com o tipo de máquina deve ser fixada.

Muitos dos problemas dos SFMs têm sido e continuam sendo encaminhados através da programação matemática [121, 201]. As ferramentas utilizadas são programação linear e não linear, programação inteira mista, programação quadrática e programação dinâmica. Um modelo utilizando programação inteira mista para analisar simultaneamente o problema de lote, carga da máquina e configuração das ferramentas é encontrado em **Chen** [48].

4.4. REDES DE PETRI

Petri expressou as ações dos elementos do sistema em um multigrafo denominado de 'grafo de rede de Petri' ou,

simplesmente, Rede de Petri [117]. Redes de Petri são adequadas para modelar atividades que necessitam de muitos recursos (tal como uma peça que requer máquinas, transporte, ferramentas), demoras (tempos de processamento, de transporte, de *setup*, etc). No entanto, até o presente, a única ferramenta disponível para analisar uma rede de Petri geral é **simulação** [202]. Por outro lado, para certas subclasses de redes de Petri, técnicas algébricas eficientes têm sido desenvolvidas para analisar e avaliar o seu desempenho. Rede de Petri é útil para investigar problemas de controle em tempo real do SFM, como para a determinação de regras de controle para ordenar operações esperando nas máquinas ou a determinação do número mínimo de *pallets* necessários para manter a taxa de produção requerida.

Uma descrição dos procedimentos para modelar um sistema de manufatura com redes de Petri é encontrado em **Argenta** [9]. Em **Caulliraux** e **Naveiro** [44] é dada uma aplicação de redes de Petri com simulação para análise do sistema Kanban em uma célula flexível de manufatura. Uma representação de um SFM através de redes de Petri e simulação pode ser visto em **Kodate, Fujii** e **Yamanoi** [117].

4.5. INTELIGÊNCIA ARTIFICAL (SISTEMAS ESPECIALISTAS)

Sistemas Especialistas têm sido aplicados em diversos áreas, tais como, diagnóstico médico, exploração de petróleo e controle da produção. Também podem ser utilizados para problemas de programação da produção.

Uma das primeiras aplicações da Inteligência Artifical na manufatura foi descrita em **Bullers et al** [34]. Um levantamento das aplicações de sistemas especialistas em Engenharia Industrial é encontrado em **Kumara et al** [119].

Simulação e Sistemas Especialistas podem ser combinados para analisar um SFM, um exemplo é encontrado em **Fahmy** [74], das seguintes formas [155]:

- Embutidos, ou o sistema especialista é embutido no modelo de simulação ou o contrário;
- Em paralelo, modelos de simulação e sistemas especialistas são projetados, desenvolvidos e implementados separadamente e interagindo em paralelo, questionando um ao outro;

- Cooperativo, nesta situação, o modelo de simulação e o sistema especialista fazem parte de um grande sistema de suporte à decisão;
- Interface (*Intelligent Front Ends - IFEs*), neste caso, o sistema especialista é encarregado de estabelecer um diálogo com o usuário com o objetivo de gerar um conjunto de instruções ou código para algum pacote de simulação particular. **Young** e **Rossi** [234] examinam o uso de conceitos de IA para melhorar o sistema de controle de um SFM, definindo os problemas de planejamento e controle e, então, examinando a tecnologia de IA que pode ser utilizada para resolvê-los.

4.6. ANÁLISE DE SISTEMAS DINÂMICOS A EVENTOS DISCRETOS (SDED)

O principal problema da implementação do SFM é o da alocação do sistema de recursos e das atividades de programação para maximizar a taxa de produção, máxima utilização de máquinas, minimizar o tempo médio de espera das peças, e/ou minimizar o tamanho da fila em cada máquina. Percebendo as estações de trabalho do SFM como nós, o sistema de movimentação de materiais como arcos conectando estes nós e as diferentes categorias de peças como entidades circulando através do sistema, o SFM será tipicamente modelado como um sistema dinâmico a eventos discretos.

SDED é dirigido para eventos discretos e cobre uma larga classe de sistemas, tais como: fluxo de materiais em linhas de produção ou montagem, fluxo de mensagens em redes de comunicações, tarefas em sistemas computacionais multi-programados. Geralmente, redes de filas podem ser caracterizadas como SDED. Existem dois enfoques para o estudo de SDED: teórico ou simulação-base, ou seja, abordagem analítica e abordagem de **simulação**. Uma terceira abordagem é a Análise de Perturbação [101]. No entanto, esta técnica ficaria melhor caracterizada como uma ferramenta de otimização da simulação [176].

4.6.1. Abordagem analítica para SDED

A abordagem analítica inclui teoria das filas e análise do valor médio e, mais recentemente, análise de perturbação. Pesquisas no desenvolvimento de modelos analíticos de SFM parecem

ter iniciado em torno de 1972/74 [35]. Os principais modelos analíticos, implementados em formas de pacotes computacionais, incluem o CAN-Q e o MVA-Q. Os modelos analíticos que têm sido desenvolvidos habilitam tratar questões relacionadas com projeto e operação do SFM. Questões como: armazenagem central versus local, carga e agrupamento de máquinas, como a capacidade global de produção é afetada pelo *mix* de peças, o número e a capacidade das máquinas, número de *pallets* e o tipo e o número de equipamentos de movimentação de materiais.

No estágio de pré-planejamento ou para verificação rápida ou quando a informação é ou difusa ou não disponível, modelos de redes de filas têm sido formulados para auxiliar no projeto, planejamento, programação e controle do SFM. Estes modelos são muito úteis na identificação de fatores que afetam o desempenho do sistema e em fornecer critérios para a tomada de decisão sobre o comportamento do sistema e interação de componentes. Modelos de filas são restritos na quantidade de detalhes que podem tratar. As hipóteses que são feitas para adequar os problemas a modelos com soluções de forma-fechada viáveis não se mostram, geralmente, apropriadas. Com as informações iniciais sobre o SFM disponíveis, o 'próximo-passo' lógico é a execução de um estudo de simulação, como normalmente praticado.

A teoria das redes de filas incluem modelos abertos e fechados e ambos são utilizados para modelar o SFM, geralmente a um nível agregado de detalhes. Estes modelos podem considerar as interações e congestões das peças competindo pelas mesmas máquinas e recursos. Muitos modelos simples de redes de filas requerem como entrada valores médios (MVA-Q, por exemplo), tal como tempo médio de processamento de uma operação numa máquina e a freqüência média de visitas a esta máquina. As saídas obtidas fornecem avaliações do desempenho de uma determinada configuração e são, também, valores médios e incluem a taxa de produção esperada no estado estático, tamanho médio de filas e utilização de máquinas.

a. Análise de redes de filas

O CAN-Q é a contribuição mais conhecida da abordagem de rede de filas para modelar SFM. Este modelo foi desenvolvido por Solberg, na Universidade de Pardue, Estados Unidos. O modelo foi

validado utilizando o SFM da empresa de tratores Caterpillar. O CAN-Q necessita do cálculo de constantes de normalização que facilmente acarretam problemas computacionais de *overflow*, mesmo para sistemas bastante pequenos [230]. Uma aplicação do Can-Q e simulação para a justificação econômica de um SFM é desenvolvida em **Co** e **Liu** [57] e para modelar o sistema de comunicação, em **Co** e **Wysk** [56]. Estes autores abordam também os riscos envolvidos em aplicar o Can-Q para analisar sistema de produção com o objetivo de orientar as pessoas envolvidas no projeto e operação de SFM.

b. Análise do valor médio

Ela é basicamente uma técnica para computar as utilizações de saídas (*throughputs*) e os tamanhos médios de filas em uma rede fechada de filas. O principal software utilizando esta técnica é o MVA-Q. É um modelo analítico de redes de filas que trata, principalmente, com os primeiros momentos (valores médios) das distribuições associadas com o problema. Ele é um algoritmo iterativo baseado na lei de **Little** (isto é, o produto do tempo médio de fluxo pela taxa de saída é igual ao número médio de ítens num sistema de filas fechado num estado estacionário). Devido ao seu procedimento heurístico (iterativo) MVA-Q pode requerer longos tempos computacionais para ser resolvido [230]. **Alberti** e **Passannanti** [5] utilizam uma abordagem de simulação e MVA-Q para estimar o custo unitário de peças manufaturas em um SFM.

c. Análise de perturbação

É um método relativamente novo que foi utilizado originalmente para estudar a localização de *buffers* de armazenagem em uma linha de produção [100] e, desde então, tem sido utilizado para estudar linhas de transferência, linhas de montagem e SFM.

A técnica é utilizada da seguinte forma. O sistema de interesse é simulado e uma amostra da trajetória é observada e dados relevantes são coletados. Um evento é perturbado. Por exemplo, um novo consumidor entra no sistema ou o tempo médio de serviço numa máquina é alterado. A perturbação é então propagada e a conseqüência de tal perturbação e outras causadas por ela são, então, rastreadas sobre o tempo ao longo da amostra inicial. O

objetivo é obter informações com uma única execução da simulação, ao invés de n replicações. Da observação de somente uma amostra, o vetor gradiente de saída é estimado. Análise de sensibilidade pode ser executada em vários parâmetros. Análise de perturbação é aplicada para responder questões do tipo 'e se'. Por exemplo, e se a média do tempo de processamento de uma peça é aumentada em 10%?. Detalhamento da teoria e aplicações aos SFM podem ser encontradas nas referências [22], [100], [101] e [109].

4.6.2. Abordagem de simulação

Entre projetistas e usuários de SFM, modelamento em simulação digital seguido por análise estatística é a mais ampla metodologia aplicada. Embora a simulação seja de base experimental (isto é, modelos de simulação são modelos descritivos), ela é usada em virtualmente todos os campos.

A análise de simulação em um SFM inclui emulação (simulação ou modelagem) física e simulação digital e gráfica.

a. Simulação física

O nível de abstração na simulação digital e gráfica, embora mais realista que em técnicas analíticas, geralmente não fornece aos investigadores condições para localizar a patologia do sistema. Ao aproximar-se a etapa de especificação e aquisição dos equipamentos, surge a necessidade de uma metodologia que projetistas e praticionistas podem usar para adquirirem experiências com os aspectos de hardware do sistema. Nos últimos anos, academias e indústrias têm recomendado o uso do modelamento físico, isto é, o uso de modelos de escala-reduzidos de CNCs, robôs e outros conponentes.

Modelamento físico consiste de equipamentos físicos de software e hardware que funcionam como módulos operando em tempo-real. Os módulos usados na emulação do sistema são, geralmente, projetados para reproduzir o ambiente operacional de um sistema de manufatura atual. Esta metodologia é recomendada por vários investigadores [58, 65, 73] e possui a vantagem de representar o sistema físico com um baixo nível de abstração.

Infelizmente, modelamento físico requer significativos investimentos em tempo, esforço e gastos para projetar e construir

(manufaturar) os componentes individuais que formam o sistema de simulação física. A metodologia, como tem sido aplicada, não considera explicitamente os aspectos administrativos das operações do sistema de manufatura.

Vários modelos icônicos têm sido desenvolvidos nos laboratórios universitários. Um simulador físico foi construído em 1982, no departamento de Engenharia Industrial da Universidade de Iowa, Estados Unidos [65]. Na escola de Engenharia Industrial da Universidade de Pardue, também nos Estados Unidos, um estudo mostrou que com componentes relativamentes baratos e dentro de um curto período de tempo, simuladores podem ser preparados e aplicados para diversos propósitos, incluindo:

- Demonstrar um complexo realista das instalações operacionais.
- Análise, experimentação e avaliação de projetos.
- Lecionar a estudantes diferentes aspectos do sistema.
- Desenvolver e testar o software de controle do sistema e o sistema de informação [58].

b. Simulação digital e gráfica

Simular um sistema de manufatura ou especialmente um SFM não é muito diferente do que simular qualquer outro sistema. Embora existam ferramentas de software específicas para modelar tais sistemas, os recursos de software utilizados cobrem uma ampla faixa e não se pode dizer que estas ferramentas específicas, denominadas **simuladores**, sejam as mais utilizadas. As razões para utilizar este ou aquele software são as mais variadas e vão desde a formação cultural do modelador até as disponibilidade dos recursos colocados a sua disposição. Depende da cultura da organização onde o modelo está sendo desenvolvido e de qual linguagem está sendo utilizada. Diversos levantamentos [52, 191, 216] comprovam que, embora os softwares de simulação estejam oferecendo cada vez mais recursos e facilidades, muitos modeladores, pode-se dizer a maioria, se valem de conjuntos de sub-rotinas (pacotes) ou linguagens de propósito geral. As opções específicas incluem fazer uso uma linguagem de simulação, um pacote orientado por dados (*data-drive*), um pré-processador ou programa gerador ou, ainda, um ambiente de suporte.

Pode-se dizer que o principal problema enfrentado por quem se decidir pela utilização da simulação para estudar um SFM é

a opção de software para codificar o modelo. Esta opção é complexa, pois a escolha do software implica em bem mais do que a codificação do modelo. Dependendo do recurso escolhido, o modelador poderá ou não se valer de animação ou de interação, terá ou não que se ocupar com a lógica do modelo, poderá ou não contar com facilidades para apresentar e analisar os resultados.

4.7. A SIMULAÇÃO NO CONTEXTO DAS TÉCNICAS DE PO

Simulação é, sem dúvida, a técnica de Pesquisa Operacional mais utilizada e é empregada de várias formas. Fazendo-se um exame das referências envolvendo simulação aplicada à problemática dos sistemas flexíveis de manufatura constata-se que ela é aplicada através de quatro abordagens.

4.7.1. Isolada

A maioria dos estudos de simulação utilizados para endereçar os problemas do SFM constituem abordagens isoladas [27, 31, 40, 41, 50, 85, 207, 224, 228], isto é, os sistemas, subsistemas e problemas são estudados em diferentes níveis de agregação, valendo-se unicamente desta técnica de pesquisa operacional.

4.7.2. Integrada com modelos analíticos

Parte dos estudos de simulação realizados são combinados com modelos analíticos. Geralmente, um modelo de programação linear [11, 173], programação inteira mista [16, 190, 204] é seguida de um estudo de simulação como forma de comprovar, validar ou, ainda, exemplicar a abordagem analítica empregada. Uma combinação igualmente freqüente é a de simulação com redes de filas. Geralmente, o modelo analítico de redes de filas CAN-Q [57, 83, 204] ou MVA-Q [5, 32] serve como uma primeira abordagem para estudar o desempenho global do sistema, enquanto que a simulação entra para modelar o sistema com mais detalhes. Verifica-se que praticamente todos os modelos analíticos utilizados para estudar os sistemas flexíveis de manufatura são seguidos de algum estudo de simulação. Um exemplo interessante destes modelos híbridos é o simulador FMSDS que combina um modelo de heurística de *layout*, no

caso uma variação do CRAFT, com um modelo analítico de filas, uma variação do CANQ-Q e um modelo de simulação.

4.7.3. Integrada com sistemas especialistas

Uma tendência que se pode observar é em direção à simulação inteligente [184, 192], ou seja, a combinação de inteligência artifical e simulação ou, mais propriamente, sistemas especialistas e simulação. As principais formas de imtegração das duas metodologias foram colocadas na secção 4.5. Um exemplo desta integração é a tese de doutorado de **Fahmy** [74].

4.7.4. Integrada ao software de controle

Esta é uma metodologia híbrida para analisar algumas questões operacionais e de projeto do SFM. A metodologia emprega um microprocessador para o controle do sistema e usa simulação para emular os sinais emanados dos centros de usinagem. Emulação de fábrica fornece um ambiente de fábrica em tempo-real para demonstração e experimentação, superando o ambiente disponível em metodologias tradicionais de simulação computacional. Com emulação as limitações de base de dados e codificação experimentadas na simulação tradicional podem ser evitadas.

Diferente de modelamento icônico, emulação de fábrica pode facilmente reconfigurar um SFM através do software, sem as dificuldades e custos de ter de redesenhar e construir os componentes de hardware que formam o sistema de simulação física [48].

Um exemplo de simulação como parte integrante do sistema de informação da fábrica para planejamento a curto e longo prazo e como ferramenta de planejamento e controle em tempo real é encontrado na referência [7]. Simulação para programação do SFM em tempo-real é vista na referência [32].

4.8. CONSIDERAÇÕES

Embora o principal propósito deste trabalho seja a análise dos instrumentos de software para codificar um modelo de simulação, convém lembrar que simular um SFM não é apenas codificar o modelo do sistema através de um determinado software.

Simulação é uma tarefa que envolve várias etapas das quais a codificação é uma delas. Pode-se dizer que num estudo típico de simulação a codificação do modelo representa 30% a 40% do trabalho total envolvido [124]. Um estudo completo de simulação envolve a modelagem do sistema, a codificação, a validação e verificação e a análise e apresentação dos resultados. A maioria dos softwares específicos para simular um sistema oferecerá auxílio, em maior ou menor grau, em uma ou mais das tarefas acima.

A tarefa de modelagem de um sistema envolve a modelagem da estrutura estática e dinâmica e da aleatoriedade do sistema. A modelagem da estrutura estática pode ser realizada basicamente de duas formas: através de um diagrama ciclo da entidade (DCE) ou através de uma rede de filas.

Na modelagem por DCE cada **entidade** do sistema (máquinas, peças, veículos, operadores, etc.) é representado como um ciclo fechado envolvendo atividades (transporte, processamento e limpeza) e filas (esperas). Vários softwares, apresentados no capítulo sete, adotam esta abordagem entre os quais o pacote **Hocus** e os programas geradores **Caps** e **Draft**. Um exemplo de modelagem de um SFM, através do DCE, é apresentado no apêndice.

Na modelagem através de redes de filas o sistema é representado por um grafo constituído por nós e arcos. Neste modelo, as entidades do sistema são vistas como consumidores (usuários dos serviços) e produtores (fornecedores de serviços). Os produtores são identificados pelos nós da rede, os consumidores como fluxos passando pelos nós e causando realizações de atividades que são identificadas pelos ramos da rede. Muitas linguagens de simulação se valem deste tipo de modelagem. Os exemplos são vistos no capítulo sete.

Independente do software utilizado, o modelador necessariamente deverá se ocupar com a modelagem estática do sistema. Em alguns softwares esta tarefa será facilitada, pois ele pode aceitar o próprio diagrama como entrada ou, então, o diagrama poderá ser introduzido através de um diálogo interativo com o usuário.

A modelagem da dinâmica do sistema envolve quatro abordagens representadas por três formas de manusear o tempo em incrementos variáveis e denominadas de visões de mundo. Esta visões de mundo consistem em identificar o evento, atividade ou

processo que ocorrerá primeiro e avançar a variável tempo (denominada de relógio da simulação) para este ponto. As abordagens de simulação utilizadas para modelar sistemas flexíveis de manufatura (ou outros sistemas discretos) são: **a duas fases** ou **evento base** e a **de Tocher** ou das **três fases,** ambas baseadas no evento e as abordagens **atividade base** e **processo base.** Dependendo da linguagem ou pacote, estas terminologias podem sofrer alterações. Com exceção das linguagens de propósito geral todos os demais softwares trazem embutidos uma ou mais abordagens da dinâmica do sistema.

A modelagem da aleatoriedade do sistema consiste em identificar as fontes de aleatoriedade, isto é, as distribuições de probabilidade adequadas e gerar valores aleatórios que simulem estas distribuições. Esta é uma tarefa que o modelador, normalmente, deverá se ocupar em maior ou menor grau, pois os softwares dificilmente fornecem todas as distribuições necessárias. Uma linguagem de programação geral, por exemplo, poderá fornecer, no máximo, um gerador de números aleatórios.

Validação e verificação são tarefas que devem ser vistas como uma parte integrante do desenvolvimento do modelo. Conceitualmente, entretanto, verificação e validação são distintas, embora normalmente conduzidas simultaneamente pelo modelador. O processo de **verificação** refere-se a comparação entre o modelo conceitual e o código computacional (programa) que implementa o modelo conceitual. Está relacionada com a correção sintática do programa e a correção da representação lógica do modelo. A **validação** refere-se a determinação de que o modelo é uma representação correta do sistema real. O auxílio que um software pode fornecer na validação e verificação de um modelo é bastante variado, mas em geral, não suficiente. Softwares que incorporam o conceito de **simulação visual interativa** [104] tendem a ser mais úteis para a validação e verificação do modelo.

A análise e apresentação dos resultados envolve técnicas estatísticas para garantir a acurária das estimativas quando a simulação é realizada com fins preditivos e para facilitar a experimentação (análise fatorial) quando com fins comparativos. Poucos softwares oferecem mais de uma técnica de estimação ou facilidades para experimentação. Em geral, é necessário a utilização de um pacote estatístico.

CAPITULO V

5. LINGUAGENS DE PROGRAMAÇÃO GERAL

5.1. INTRODUÇÃO

Simulação, entre outras técnicas de Pesquisa Operacional, necessita, por sua natureza, do projeto e da codificação de uma peça original de software. Existem, pelo menos, três razões para a programação [154]:

- Quando for notado que um software pode ser produzido a um custo mais baixo do que um software comercial disponível,
- Existem vantagens em se ter um software sob medida. Por exemplo, se um modelo é rodado diariamente, pode-se economizar tempo computacional, através de um programa específico para o problema e seus dados associados e
- É necessário um software mais simples que um comercialmente disponível.

A escolha a ser enfrentada por um modelador que deseja escrever seu próprio programa é um problema decisório complexo face ao número de opções envolvidas, recursos oferecidos e a variedade de abordagens seguidas pelos diferentes instrumentos de software.

O recursos de software são classificados conforme sua sofisticação. Por sofisticação entende-se, não somente a amigabilidade do software em geral, mas mais especificamente a extensão com que ele libera o usuário da necessidade de programação [39]. As opções para codificar um modelo de simulação, isto é, obter um programa computacional e que requerem conhecimento de programação (em diferentes níveis) incluem:

1. Linguagens de programação geral (LPG);
2. Conjuntos de sub-rotinas (pacotes).
3. Linguagens de simulação (LS);

Embora as três envolvam programação, a segunda e a terceira são opções de 'alto-nível'. Na primeira, o usuário deve se envolver em todos os detalhes, isto é, com a lógica do programa computacional. Com a terceira, ilustrada pela linguagem **Simula**, por exemplo, o vocabulário, as funções e os procedimentos da linguagem

são projetados especialmente para satisfazer as necessidades da modelagem em simulação. Entre as opções dois e três, vários níveis de sofisticação são observados. No entanto, com outras ferramentas o usuário pode não precisar programar. Estas opções são:

4. Pacotes orientados por dados (*data-driven*),
   4.1. Modelos genéricos de manufatura,
   4.2. Modelos genéricos para manufatura flexível,
5. Programas geradores e pré-processadores,
6. Ambientes de suporte.

No grupo quatro é suficiente fornecer os dados para especificar entidades, atividades, filas, etc. A lógica é especificada não como um programa computacional, mas em termos de um DCE ou equivalentes, também na forma de perguntas/respostas ou, ainda, como entradas em campos de dados.

No grupo 4.1, a terminologia é orientada para a manufatura, ao passo que, o grupo 4.2 possui termos de referência mais limitados e modelam tipos específicos de sistemas de manufatura, tais como os SFM. Neste caso, a lógica já foi determinada por quem escreveu o modelo genérico, o usuário deve somente indicar os componentes presentes e o fluxo destes componentes através do sistema. Com modelos genéricos de manufatura isto normalmente é feito, mas com modelos específicos para SFM, o usuário, em geral, deve apenas selecionar regras de decisão do fluxo de trabalho, de um menu apresentado a ele.

Os recursos do grupo cinco geram o código do programa automaticamente, como o **Caps**, por exemplo. O grupo seis vai além do conceito de automatizar a codificação do modelo, para incorporar auxílio à modelagem, experimentação e análise dos resultados da simulação, fornecendo, ainda, animação e interação com o usuário, proporcionando, desta forma, quase um estudo completo de simulação.

Estas seis abordagens ilustram o espectro dos níveis de sofisticação, a começar da primeira, em que cada detalhe do modelo deve ser definido e programado pelo modelador, para a última, em que nenhum detalhe da lógica do modelo necessita ser escrito e, somente uns poucos detalhes são deixados para a escolha do usuário e que fornecem, também, auxílio nas demais etapas de um estudo de simulação.

O propósito deste capítulo é examinar as opções de

software não específicos para simular um SFM representadas pelas linguagens de propósito geral.

5.2. FATORES A CONSIDERAR NA SELEÇÃO DE UMA LINGUAGEM DE PROPÓSITO GERAL PARA SIMULAR UM SFM.

É importante entender porque simulação computacional discreta é possível de qualquer modo. A possibilidade é baseada no uso de um computador como uma máquina lógica melhor do que como uma máquina calculadora. Muita Pesquisa Operacional é baseada na habilidade dos computadores executarem cálculos rapidamente e de forma acurada. Para simulação discreta, velocidade e acurácia são importantes, mas a característica fundamental é a habilidade do computador de executar tarefas lógicas. A fim de ter utilidade para simulação discreta, uma linguagem de programação deve facilitar a expressão destas interações lógicas.

Existem três características que uma linguagem deve satisfazer para que isto ocorra [164]:

1. Permitir a nomeação de variáveis e outras entidades.

A linguagem deve permitir a utilização de nomes sugestivos para variáveis e outras entidades.

2. Declarações poderosas.

Mais significativo é a facilidade com que declarações lógicas podem ser feitas. A forma típica de expressão para interação lógica em uma simulação discreta é a seguinte:

Se <condição> então <ações> senão <outras ações>.

A declaração *case*, do Pascal, pode tornar o programa de simulação mais legível e eficiente, pois ela evita uma longa lista de declarações, *if ... then ... else*, em cascata.

Para expressar de forma conveniente ações e testes repetitivos, podem ser utilizadas as declarações *loop ...do* , *for* e *go to*. Contudo, existem modos melhores, incorporados por muitas linguagens de programação recentes, de se alcançar o mesmo objetivo. São declarações como:

(a) *While*

Encontrada em simulação igualmente na seguinte forma: *while* <recursos disponíveis> comprometa recursos.

(b) *Repeat ... until*

Encontrada em simulação na seguinte forma *Repeat* <ações> *until* <condições>. Estas declarações conduzem à programas que são fáceis de ler e convenientes de escrever.

(3) Tipos de dados variados.

A linguagem deve fornecer a possibilidade do usuário definir seus próprios tipos de dados. Dados do tipo 'registro' também são importantes, bem como gerenciamento dinâmico de memória através de ponteiros. Esta característica é fundamental, uma vez que desta forma pode-se aumentar consideravelmente o número de entidades temporárias no modelo, pois à medida que a entidade passa pelo modelo seu espaço de memória é liberado.

Além destas três características existem outras que são importantes, pois facilitam o trabalho do programador [164]:

1. Rotinas para modelar a aleatoridade do sistema

Sistemas discretos incluem algum tipo de aleatoriedade, portanto, devem existir rotinas para a geração: de números aleatórios, das principais distribuições de probabilidade e para distribuições empíricas.

2. Rotinas para coletar estatísticas.

Rotinas que coletem estatísticas automaticamente, produzam histogramas, traçem gráficos e estimem momentos para as variáveis de saída do sistema.

3. Rotinas de manuseio de filas.

Rotinas para manter listas (ponteiros) permitindo a entrada e a saída de novas entidades destas listas, de acordo com vários critérios (disciplinas).

4. Rotinas de aplicações específicas

São rotinas específicas dos SFM como os centros de: usinagem, montagem, limpeza, controle e robôs, veículos automatizados, esteiras, áreas de armazenagem, etc.

Programas de simulação são, normalmente, longos e, portanto, deve-se aplicar tanto quanto possível os conceitos de Engenharia de Software, como projeto *top-down* e modularidade. O

conceito de linguagem hoje é suficientemente elástico e na realidade envolve todo um ambiente que deve fornecer estruturas para escrever, corrigir, alterar, manter, adicionar e suprimir partes do programa computacional. Desta forma, além dos critérios acima, para se optar por uma ou outra linguagem de programação, os seguintes critérios devem ser observados [80]:

1. Editor de texto

Este componente do sistema é utilizado para entrar o programa e outros dados no computador. O método de entrada comum é por edição de texto interativa, que torna fácil a correção do texto do programa.

2. Macroprocessadores

Macroprocessador é um tradutor simples que pode substituir seqüências específicas em um documento fonte com seqüências alvos projetadas. Ele pode ser utilizado para suprir deficiências da linguagem, aumentar a legibilidade ou substituir conjuntos de sentenças da linguagem.

3. Interpretador/compilador

Um interpretador é usado para executar um programa fonte. Ele pode diagnosticar erros melhor do que um compilador e é desta forma mais apropriado para a correção do programa. Por outro lado, um interpretador é menos eficiente no tempo de execução. Seria desejável ter um interpretador e um compilador que aceitassem a mesma linguagem de modo que o interpretador fosse utilizado para diagnóstico e o compilador para produção. O sistema **Interlisp** inclui tal combinação para a linguagem **Lisp.**

4. Sistema de arquivos

Um programa de simulação é normalmente longo e um sistema de arquivos que forneçam facilidades para combinar, renomear, destruir, criar e fundir estes arquivos deve existir para armazenar dados e/ou programas em código

5. Editor de ligação ou *linker*

Um *linker* toma vários módulos compilados independentementes e junta-os em um único. A disponibilidade de um

linker apresenta muitas vantagens. Permite que vários programadores trabalhem em diferentes módulos ao mesmo tempo. Permite que um módulo seja compilado uma única vez e utilizado em vários sistemas diferentes. Se um erro é encontrado em um módulo apenas ele necessitará ser recompilado.

6. Tipos fortes

Uma linguagem é **fortemente tipada** se: (i) todo objeto na linguagem pertence exatamente a um tipo; (ii) a troca de tipos ocorre convertendo um valor de um tipo para outro - a troca não ocorre pela consideração da representação de um valor como um tipo diferente e (iii) cada tipo de objeto deve ser explicitamente declarado antes de ser utilizado.

Um ponto de vista atual é que uma linguagem fortemente tipada ajuda na construção de programas claros e legíveis. Erros comuns de programação (como o nome incorreto de variáveis ou valores ilegalmente atribuídos) podem ser detectados durante a compilação de uma linguagem fortemente tipada.

7. Eficiência da implementação da linguagem

Em se escolhendo uma linguagem, é essencial entender a diferença entre a linguagem e uma implementação específica desta linguagem. Um compilador ou interpretador ruim podem tornar a linguagem praticamente inútil e, também, uma linguagem pode ser muito difícil de implementar num determinado computador (configuração), embora possa ter um excelente compilador (interpretador) em outro. Lembrando, ainda, que programas de simulação são normalmente longos, o que torna a escolha do compilador ou interpretador ou, ainda, do computador cruciais. Além disso, de nada adianta ter uma linguagem excelente e um compilador eficiente se for necessário o dobro de memória do que a do computador disponível ou, então, de uma resolução gráfica acima da do monitor em uso.

5.3. LINGUAGENS DE PROGRAMAÇÃO GERAL

A seguir são colocadas as características das principais linguagens de programação em uso e que são utilizadas em maior ou menor grau para codificar modelos de simulação de sistemas

flexíveis de manufatura.

5.3.1. Fortran

O desenvolvimento do Fortran é paralelo ou mesmo equivalente ao desenvolvimento da própria computação.

O projeto do Fortran é centrado no seu principal objetivo, que é a execução eficiente, e pode ser implementado em praticamente qualquer computador.

Pelo seu amplo uso, o Fortran extende-se para muito além do projeto inicial de uma linguagem científica. O crescimento maior se deu pela adição de pacotes de sub-rotinas. Um programa Fortran consiste de um programa principal e um conjunto de sub-rotinas, cada qual compilada separadamente das demais, com os programas traduzidos ligados na forma final durante a execução.

Em virtude do Fortran ter sido a primeira linguagem de alto nível a ter largo uso, a maioria das linguagens de simulação são baseadas em Fortran de uma forma ou de outra. Esta tendência só foi revertida nos anos 80, com a grande utilização do Pascal e do C.

A proeminência do Fortran na Pesquisa Operacional (e particularmente em simulação) é devido, principalmente, às seguintes razões [154]:

a. Ele é historicamente a linguagem da comunidade científica. Cientistas têm investido tempo e esforço para se familiarizarem com Fortran e precisam de boas razões para substituí-lo.

b. Ele tem um bom suporte dos principais fabricantes de computadores. Como resultado, compiladores são razoavelmente vigorosos. A implementação é boa e uma documentação extensiva está diponível.

c. A portabilidade de softwares codificados em Fortran é comparativamente boa.

d. Está equipado para compilação separada e acesso à bibliotecas de sub-rotinas.

e. Existe a necessidade de manter o legado dos softwares existentes escritos em Fortran.

A principal virtude do Fortran é a grande disponibilidade de compiladores eficientes e livres de erros. Eles produzem códigos objeto bem otimizados e são totalmente suportados

por seus vendedores. Isto não é necessariamente verdadeiro para qualquer linguagem de simulação discreta e poderá ser difícil obter suporte técnico apropriado se um pequeno grupo de especialistas na linguagem está em um lugar distante. Em contraste, especialistas em Fortran podem ser encontrados em qualquer lugar [164].

Fortran foi ultrapassada pelas linguagens bem estruturadas. Ela fornece poucos recursos de controle e falta recursão (isto é, onde uma rotina pode chamar a si própria). Mesmo entusiastas do Fortran reconhecem que a linguagem é deficiente em tipos de dados. Fortran possui somente os seguintes: reais, com precisão simples e dupla, inteiros, lógicos, arranjos de reais, inteiros e lógicos e complexos. A versão 77 da linguagem melhorou em parte sua capacidade, por exemplo, ela incluiu variáveis do tipo 'caracter'e a declaração *if ... then ... else*.

Muitas instituições utilizam as vantagens do Fortran, enquanto evitam algumas de suas desvantagens pela utilização de um pré-processador, processando uma linguagem bem estruturada não-Fortran. O código Fortran é, então, compilado normalmente. O custo do pré-processamento é presumivelmente compensado pelo menor tempo de desenvolvimento do programa, ganho em utilizar uma linguagem bem estruturada. Existem numerosos (de fato centenas) pré-processadores Fortran, sendo que o mais conhecido é provavelmente o **RatFor** de **Kernighan e Plauger** [154].

Aplicações do Fortran para simular sistemas flexíveis de manufatura estão exemplificadas nas referências [32], [161], [194], [207] e [209]. Do Fortran com Gpss na referência [11] e em [133] do Fortran com Ecsl.

5.3.2. Pascal

O Pascal herdou a forma sintática geral e o *layout* dos blocos estruturados, a elegância e a simplicidade do Algol 60, que desempenhou nos anos 60 um importante papel influenciando outras linguagens e estudos teóricos de sintaxe e semântica. O mesmo papel foi desempenhado pelo Pascal nos anos 70, influenciando por sua vez linguagens tais como: Ada, Modula-2 e Pascal Concorrente.

Em Pascal, programas são estruturados em árvores com funções e procedimentos como módulos. Modularidade é uma

característica essencial da linguagem e, desta forma, de programas Pascal. Isto promove projeto *top-down* e permite o desenvolvimento individual de módulos testáveis. Por outro lado, estrutura de árvore aninhadas não são permitidas em Fortran, somente funções individuais e sub-rotinas. A modularidade em Fortran deve ser uma opção consciente do programador, ao invés de uma consequência do uso da linguagem. Desta forma, Pascal parece melhor do que Fortran para o desenvolvimento de programas modulares [163].

A parte mais inovadora do projeto é o tratamento do tipo de dados. Um grande número de tipos de dados está disponível: inteiros, reais, caracteres, enumerados, lógicos, arranjos, sub-faixas, registros, arquivos texto e seqüenciais e conjuntos. Há, ainda, ponteiros e a possibilidade do usuário definir seus próprios tipos de dados a partir dos oferecidos pela linguagem [166].

Tipos enumerados, tipos sub-faixas e construídos serão considerados aqui, do ponto de vista da simulação, de um sistema flexível de manufatura.

(a) Tipos enumerados

Permitem ao programador declarar seus próprios tipos de dados como escalares. Por exemplo:

Type máquina = (torno, fresa, furadeira);

Var recurso : máquina;

Isto declara 'recurso', como sendo uma variável do tipo 'máquina', que só pode assumir os valores acima definidos. Os operadores relacionais: =, <>, >, <, ≤ e ≥, podem ser aplicados a estes valores. No caso: torno < fresa < furadeira

'Recurso' também pode ser usado como uma variável de controle de repetição, como por exemplo:

For recurso := torno *to* furadeira *do*

Estes tipos são um bom recurso para evitar a utilização de variáveis simbólicas.

(b) Tipos sub-faixas

Oferecem uma maneira útil de evitar situações onde variáveis podem mas não devem ir além de um certo limite. O programador de simulação pode ser vítima deste problema. Por exemplo, se a capacidade máxima de uma máquina é de 300 unidades por hora, então, o programador pode definir o seguinte tipo sub-faixa:

Type npeças = (0..300);

Var capacidade = npeças;

Desta forma, a variável capacidade é declarada como sendo do tipo 'npeças', isto é, um inteiro na faixa de 0 a 300. O compilador pode, então, fazer a checagem da variável capacidade para ver se está fora da faixa, quando o programa é executado.

(c) Construídos

Pascal permite vários tipos construídos, isto é, tipos feitos de outros tipos. Os mais familiares para entusiastas do Fortran ou Basic é o arranjo (*array*). Por exemplo:

```
Type npeças = (0..300);
     Máquina = (torno, fresa, furadeira);
     Capmat = array[1..10, 1..7] of npeças;
     Maqmat = array[1..5] of máquina;
Var Cap_bloco : capmat;
    Bloco_maq : maqmat;
```

Desta forma, 'capmat' é declarado como um arranjo bi-dimensional do tipo 'npeças'. 'Maqmat' é declarado como um vetor (arranjo unidimensional) do tipo 'máquina'. 'Cap_bloco' é declarado como uma variável do tipo 'maqmat'. Variáveis destes tipos podem ser utilizadas, então, no programa. O que é particularmente conveniente é que elas podem ser passadas para os procedimentos como parâmetros. Arranjos podem ser declarados como sendo de qualquer tipo, quer embutidos, enumerados ou construídos.

Outro tipo construído permitido em Pascal e algumas outras linguagens é o tipo de dados conjunto (*set*), que pode ser usado para permitir testes nos estados do sistema muito convenientes. Por exemplo:

Type estado_possível = (trabalhando, parado, quebrado);

Var estado : estado_possível;

O programa, então, pode incluir:

If estado *in* [quebrado, parado] *then* ...;

Isto torna o programa muito conveniente e legível.

Menos familiar para o programador, Fortran é a idéia de registro (*record*) como utilizado em Pascal e outras linguagens. Este tipo construído permite uma estrutura composta de vários tipos diferentes. Por exemplo, suponhamos que se quer informações ligadas sobre tempo, disponibilidade, estado e atividades de uma

entidade. Pode ser declarado, então, o seguinte registro:

```
Type máquina = (torno, fresa, furadeira);
     Car25 = array[1..20] of char;
     Atividade = (partida, parado, reparo);
     Estados_pos = (executando, OK, vazio, limpeza);
     Detalhes = record;
     Nome = car25;
     Disponível : boolean;
     Tempo_célula : integer;
     Prox_ato, Ult_ato : atividade;
     Estados : estados-pos;
End;
```

Se o número de entidades no programa é pequeno e fixado, então estas entidades registros podem ser combinadas em um arranjo. Supondo os tipos acima:

```
Var det : array[máquina] of detalhes;
```

Isto permite referências muito convenientes de informações para cada entidade. Por exemplo, se desejamos atualizar a informação sobre o 'torno' então:

```
With det[torno] do if estado = OK then
Begin
    Tempo_célula := 20;
    Prox_ato := partida;
    Estado := executando;
    Disponível := false;
End;
```

Outro tipo desconhecido por programadores Fortran é o ponteiro (*pointer*). Ele é usado para permitir variáveis dinâmicas, isto é, aquelas que são criadas e destruídas enquanto o programa está sendo executado. Para simulação, este é um tipo de dado muito interessante. Por exemplo:

```
Type Car20 = array[1..20] of char;
Atividade = (chegada, posicionada, trabalhada, retirada)
Pro_estado = (estoque, espera, retrabalho);
Ent_ponteiro = ^Ent_detalhes;
Ent_detalhes = record;
    Nome : car20;
    Disponível : boolean;
    P_ação, U_ação : atividade;
```

```
        Estado : pro_estado;
        Prox_ent : ent_ponteiro;
    End;
    Var peça : ent_ponteiro;
```

Este exemplo emprega um registro semelhante ao anterior, exceto pela adição de um novo campo, que é chamado 'prox_ent' e é do tipo 'ent-ponteiro', que é previamente declarado como um ponteiro para 'ent_detalhes'. Por fim, uma variável 'peça' é declarada como sendo do tipo 'ent_ponteiro'

Esta variável é, desta forma, um ponteiro para qualquer registo 'peça' especificado. Tais registros podem ser criados e destruídos à medida que a simulação prossegue. Quando registros 'peça' são criados, eles são ligadas a uma 'peça' anterior usando o campo 'prox_ent', que é ele próprio um apontador de registro 'peça'. Desta forma, registros podem ser armazenados como uma lista encadeada para a frente e para trás, evitando a declaração prévia do número de peças no sistema. Tal declaração seria necessária para a dimensão de um arranjo (*array*) se ele fosse utilizado como uma estrutura agregada de registros. Listas semelhantes podem ser usadas para representar filas.

Deve-se salientar que listas deste tipo não são tão convenientes para processsamento quanto um arranjo. No arranjo é possível ir direto a qualquer elemento particular, enquanto num ponteiro ele deve ser procurado. Entretanto, se existem muitas entidades temporárias, então variáveis ponteiros serão muito úteis.

Pascal está disponível em micros, através da versão padrão para micros que é o UCSD Pascal, desenvolvida na Universidade da Califórnia em San Diego. Outra opção é o Turbo Pascal da Borland, uma extensão do Pascal, que combina um editor, um compilador e um depurador criando um ambiente de desenvolvimento de software muito produtivo, incluindo alta capacidade gráfica. A última versão, o turbo 6.0, inclui programação orientada para objetos.

Cheng [49] aplica a linguagem Pascal para avaliar o desempenho de SFM alternativos e coloca "a linguagem foi utilizada porque é moderna e eficiente e compiladores estão disponíveis em mini e microcomputadores. Além disso a linguagem possui mais usuários do que linguagens especiais de simulação como Q-Gert e

Gpss, que por sua vez requerem compiladores especiais que são mais caros e menos portáveis do que os da linguagem Pascal".

5.3.3. C

Compiladores C estão disponíveis em uma grande variedade de computadores, desde micros até mainframes. Um programa escrito em C pode ser rodado num micro ou mainframe, desde que se tenha um compilador C padrão. Desta forma, uma das principais carcterísticas do C é a sua portabilidade.

Esta portabilidade e flexibilidade aliada a características especiais tornam C uma linguagem ideal para simulação. Algumas capacidades especiais que favorecem a simulação são: especificação de atributos, tipos de dados ponteiro e alocação dinâmica de memória. C vem sendo ativamente estendida e refinada. Uma nova versão de C, denominada C++, desenvolvida nos laboratórios da Bell, fornece capacidades adicionais na forma de um pré-processador que permite ao usuário extender a linguagem para a criação de novos tipos de dados e operadores. Para simulação, isto significa que se pode criar operadores que desempenharão as funções muitas vezes necessárias em um modelo de simulação.

Uma das características do pré-processador é permitir ao programador representar qualquer texto *string* com outro texto *string* através do comando "*#define*". Pela utilização deste comando o usuário pode substituir constantes, expressões, ou mesmo sentenças inteiras, com nomes que são mais significativos para a aplicação em mente.

Uma outra característica do C é a sua habilidade para lidar com ponteiros, aumentando a eficiência na manipulação de estruturas complexas. Em simulação discreta faz-se uso, muitas vezes, de funções que desempenham tarefas envolvendo entidades (programação, arquivamento, etc.). Pela utilização de uma estrutura ponteiro, acesso a todos os atributos de uma entidade podem ser passados para estas funções com um único argumento.

Muitas linguagens de programação modernas fornecem facilidade para a alocação dinâmica de memória. C não é exceção. Através do uso de funções, o programador pode alocar e liberar blocos de memória enquanto o programa está sendo executado. Isto

traz vantagens óbvias para a simulação. Quando o modelo de simulação é construído, não se tem, na maioria das vezes, um conhecimento preciso dos congestionamentos do sistema (isto é, do número de entidades no sistema simultaneamente). Em linguagens que fornecem somente alocação estática de memória, notadamente Fortran, esta situação causa ao programador a impossibilidade de armazenar toda a lista de entidades ou o risco de abortar a simulação devido a falta de memória. O problema é duplicado pelo fato de que para alterar a quantidade de memória alocada é necessário recompilar um ou mais módulos e religar (*link*) o sistema inteiro.

Uma linguagem que tem a capacidade de requisitar e liberar memória dinamicamente para armazenar registros temporários de simulação, leva uma vantagem adicional em eficiência. Linguagens como Pascal e C fornecem ferramentas superiores para o desenvolvimento de softwares de simulação [189].

Desde que manipulação de listas prevalece em simulação, um dos serviços que deve ser fornecido por um pacote de simulação discreta deve ter funções para manipular listas tais como: inserir, remover e encontrar entidades em uma lista. Deve ter ainda uma função para programar eventos. C possui algumas capacidades que a tornam apropriada para escrever funções que executam manipulação de listas. Entre elas a mais poderosa é a já mencionada ponteiro. Um outro problema de simulação que C resolve através da habilidade de criar novos tipos de dados são os múltiplos controles de tempo, uma vez que alguns tipos de simulação requerem o tempo como variável inteira, enquanto outros como variável real.

C fornece, ainda, um operador de atribuição que é muito próximo ao conjunto de instruções de máquina. O resultado é um código compilado compacto e eficiente, inteiramente apropriado ao objetivo principal de um modelo de simulação computacional, que é a coleta de estatísticas de desempenho do sistema sendo simulado.

Por fim, C fornece um conjunto variado de operadores sobre *bits* para acelerar o processo de divisão, uma das operações computacionais mais lentas, necessária à geração de números aleatórios. Através de diretivas de compilação pode-se escrever funções de geração de números randômicos portáveis para qualquer máquina com palavras de tamanho 16, 32 ou 36 bits [88].

C é uma linguagem simples e poderosa que está se tornando a mais importante linguagem de aplicações industriais e para o desenvolvimento do software. Pelo projeto do C torna-se fácil incorporar planejamento *top-down*, programação estruturada e projeto modular em modelos de simulação.

Em resumo, as capacidades do C que são mais pertinentes para simulação são [189]:

1. Gerenciamento dinâmico da memória,
2. Suporte para manejar dados utilizando ponteiros,
3. Suporte para a construção de software modular,
4. Suporte flexível e padrão de I/O,
5. Portabilidade e
6. Disponibilidade de bons compiladores e ambientes de desenvolvimento [189].

A principal desvantagem da linguagem é que a ausência de tipos fortes permite a escrita de programas obscuros e que podem, dentro de um período relativamente curto de tempo, tornarem-se impenetráveis para o autor e principalmente para outros.

Para **Crookes** [164], não existe dúvida que o uso sério do C impõe a necessidade de uma disciplina pessoal de programação que Pascal é projetada para impor ao usuário. Devido à sua grande flexibilidade, erros podem ser cometidos e resultarem em C legal, sendo somente detectados pelo comportamento fora do esperado do programa em tempo de execução.

5.3.4. Ada

O projeto do Pascal foi o ponto de partida para o do Ada, mas o resultado difere em muitos aspectos do Pascal. Ada é uma linguagem mais extensa e mais complexa do que Pascal. Inclui conjuntos maiores de características que não possuem análogos em Pascal, em particular, execução concorrente, controle em tempo real de tarefas, manuseio de exceções e tipos abstratos de dados, apresentando, no entanto, tipos fortes tal qual Pascal.

Ada foi planejada para suportar a construção de grandes programas por equipes de programadores. Um programa Ada é, normalmente, construído como uma coleção de 'componentes de software' chamados 'pacotes'. O pacote é o mecanismo da linguagem Ada que permite recursividade. Tais pacotes devem estar

disponíveis para o programador Ada através de uma biblioteca de pacotes e a primeira tarefa do programador Ada é fazer a construção do programa através da combinação dos pacotes existentes. Muito da natureza especial do Ada e da programação nesta linguagem vem da sua ênfase na construção de pacotes utilizando pacotes.

Pacotes permitem a definição e uso de bibliotecas de rotinas comuns compiladas separadamente. O conceito é semelhante ao de unidade (*unit*) do Pascal e o de módulo (*module*) do Modula-2. No entanto, um pacote em Ada pode ser genérico, isto é, diferentes exemplos de pacotes podem ser gerados. Por exemplo, é possível escrever um pacote genérico de processamento de listas, onde diferentes exemplos do pacote podem manipular ítems da lista de diferentes tipos e trabalhar com diferentes quantias de armazenagem. A declaração de um pacote deve especificar sua interface com o mundo exterior de forma que não contrarie sintaxe fortemente tipada da linguagem Ada.

Os principais procedimentos da linguagem aplicáveis à construção de programas de simulação discreta são os pacotes e as tarefas, onde uma tarefa é uma unidade do programa que é executada concorrentemente (em paralelo, pelo menos conceitualmente) com outras tarefas.

O caminho padrão para representar entidades particulares em Pascal é por registros ou arranjos (*arrays*) de registros. Componentes dos registros representam os atributos de uma entidade. Esta representação também pode ser utilizada em Ada.

Ponteiros são o caminho natural para representar entidades temporárias, uma vez que eles podem ser criados, movidos através da simulação e descartados. Normalmente, deseja-se manter listas de entidades. Listas encadeadas podem ser implementadas pela alocação de ponteiros em cada entidade e criando um tipo registro para representar o início da fila. Pode-se construir rotinas que inserem e retiram ítems da fila. Em Pascal, o programador deve declarar a estrutura de dados e as rotinas de manutenção para filas separadamente. Em Ada, o programador pode declarar um pacote fila que inclui tanto a estrutura de dados quanto as rotinas de manutenção. Em Ada, como tarefas são executadas em paralelo, vários processos podem estar ativos simultaneamente. Desta forma a linguagem Ada permite uma abordagem

processo orientada melhor do que evento orientada, prestando-se à construção de programas de grandes sistemas com eficiência e confibialidade [33].

5.3.5. Modula-2

Modula-2 é um descendente direto do Pascal e pode-se imaginá-la como uma linguagem melhor e mais poderosa. Desta forma, esta linguagem apresenta pelo menos as mesmas facilidades do Pascal para se codificar um modelo de simulação.

Além das características do Pascal, Modula-2 oferece abstração de dados, compilação separada e co-rotinas. Abstração de dados é o conceito de que um tipo de dado inclui não somente um conjunto de objetos de dados mas também um conjunto de operações que manipulam estes dados. Tipos abstratos de dados permitem o desenvolvimento de um algoritmo utilizando-se de objetos naturais que não são, necessariamente, os primitivos da linguagem. Por exemplo, um tipo de dado abstrato está embutido na definição de uma 'fila/fifo' de inteiros e as operações 'entrar' e 'sair' associadas a ela. O modelador pode definir um objeto deste tipo e utilizá-lo através de suas operações, mas não precisa saber se a fila foi implementada usando uma união de elementos, um arranjo (*array*) ou outra estrutura qualquer.

Um programa em Modula-2 é, normalmente, constituído de uma série de módulos, onde cada um é uma coleção de procedimentos e estrutura de dados relacionados que implementam uma seção bem-definida do programa total. Um módulo que usa um outro módulo é denominado um cliente e o que fornece serviços, de servidor, sendo que qualquer módulo pode ser tanto cliente quanto servidor.

A compilação dos módulos pode ser feita separadamente e armazenadas em código de máquina, para utilização posterior. Cada módulo deve especificar se está usando outro módulo, sendo que esta informação torna possível a ligação (*link*) automática no ambiente Modula-2.

Modula-2 tem suporte embutido para multitarefa. E a bibilioteca da Modula-2 inclui um avançado programador divisor-de-tempo que torna muito fácil implementar processos concorrentes e simulação.

5.3.6. Basic

Assemelha-se ao Fortran na estrutura geral, mas possui uma sintaxe mais simples com maiores opções gráficas.

Para usuários de micros as vantagens do Basic são:

1. Fabricantes de micros a tem desenvolvido extensivamente.
2. Ela está normalmente disponível na *ROM*.
3. Muitas implementações do Basic não fazem distinção entre suas estruturas de edição e as de execução tornando-a, assim, fácil de se usar.

Existem, entretanto, três desvantagens consideráveis desta linguagem:

1. É uma linguagem mal estruturada, simples e com estruturas de dados limitadas e um inadequado manuseio de arquivos.
2. Um mecanismo de procedimentos pobres e carência de estruturas para compilação separada tornam o desenvolvimento de grandes programas difíceis de manejar.
3. Não é portável, pois os fabricantes de micros a alteraram em tal extenção que o conceito de linguagem é inapropriado.

Aplicações desta linguagem em simulação existem, porém em pequeno número. Um exemplo é MSBASIM, de **Pidd** [163], escrito em Basic da Microsoft, baseado numa versão de **Crookes**, da Universidade de Lancaster, Inglaterra. O próprio autor do software fornece uma versão do mesmo programa em Turbo Pascal da Borland. **Haddock** [94] implementou um pré-processador Basic, para a linguagem Siman, com o propósito de projetar e controlar um SFM.

5.3.7. PL/I

É uma linguagem multi-objetivo, e extensa. Seu principal objetivo de projeto era suceder ao Fortran fornecendo maiores facilidades de estrutura de dados com ambiente operacional mais sofisticado e com aplicações mais amplas que Fortran.

Baseia-se em grande parte no Fortran, Algol e Cobol. É um híbrido entre Fortran e Cobol, com estruturas adicionais para manipulação de baixo nível de *bits* e endereços. O aspecto mais interessante de sua construção foi a tentativa dos projetistas de manter o eqüilíbrio entre dois objetivos conflitantes, a generalidade e a flexibilidade, sem perda da eficiência de

execução e facilidade de uso por programadores não-sofisticados, e perda do controle detalhado por programadores mais experientes. A resultante foi uma linguagem extensa e complexa. Para torná-la utilizável por programadores menos experientes, a linguagem adota a filosofia de extensivas declarações *default*.

Uma extensão da linguagem para utilização em simulação é fornecida pela SIMPL/I.

5.3.8. Lisp

Para se ter uma idéia das estruturas que esta linguagem fornece é necessário entender o conceito básico de lista. Ao invés de se armazenar dados seqüencialmente na memória, cada ítem contém não só os dados, mas o endereço do próximo dado em seqüência lógica. Estes ítems são conhecidos como ponteiros (*pointer*) e são adotados hoje pela maioria das linguagens modernas.

Talvez a mais conhecida e utilizada linguagem de processamento de lista seja o sistema **Lisp**. Este sistema representa uma abordagem diferente para o assunto. Lisp tem sua base fundamental em aspectos da teoria das funções matemáticas e recursivas, que a torna virtualmente diferente de qualquer outra linguagem. Isto significa que o usuário tende a escrever funções em Lisp e seus programas consistem de intruções para avaliar as funções de modo a produzir os resultados desejados.

Processamento de listas é a base dos algoritmos Lisp, embora números e caracteres possam também ser limitadamente manipulados.

Um exemplo de simulação em Lisp é o sistema **Ross**. Este sistema foi projetado para aumentar a inteligibilidade, a capacidade de modificação e o desempennho de simulações extensas. A linguagem Lisp é utilizada para construir um simulador como em **Wu** e **Wysk** [231], que a utilizam para simular alternativas de programação e controle do SFM. É utilizada, ainda, como interface para automatizar a modelagem em uma linguagem de simulação (Gpss) com o objetivo de desenvolver um sistema automático de simuladores de manufatura como em **Schroer** e **Tseng** [184]. Lisp também é utilizada para escrever sistemas especialistas (SE) que utilizam simulação para avaliar SFM como em **Fahmy** [74].

5.3.9. Prolog

A maioria dos trabalhos correntes em simulação é feito em linguagens de propósitos gerais tais como: Fortran, Pascal ou C ou em linguagens de simulação como Gpss ou Simscript. A característica comum destas linguagens é a sua natureza imperativa, ou seja, impõe como o computador deve resolver o problema desejado. O programa especifica explicitamente os passos que devem ser seguidos para se obter a solução do problema considerado. Isto significa dizer que o programador é forçado a transcrever o problema em um esquema 'próprio da máquina'.

Prolog implica em uma mudança radical na maneira de encarar o problema: informa-se à máquina o que é para ser feito e que fatos e regras relacionam-se ao problema. É somente necessário descrever o problema em termos de sentenças e regras afetando os objetos em questão. Se a descrição do problema for suficientemente precisa o problema pode ser resolvido. Desta forma, Prolog pode ser melhor caracterizada como uma linguagem descritiva.

Prolog é uma linguagem baseada em lógica simbólica, projetada para representar e usar fatos sobre um campo do conhecimento. Fatos são representados por um conjunto de relações que descrevem as propriedades dos objetos e suas interações.

Um programa Prolog é uma base de dados composta de fatos e regras. Os fatos representam as relações lógicas entre objetos e atributos. Qualquer sentença que se escreve em Prolog torna-se parte da base de dados. Declarações podem ser fatos, perguntas ou comandos.

Para simulação, a orientação do Prolog pode ajudar usuários a encontrar mais facilmente o modelo adequado. O processo típico de procura que leva a um modelo não é bem suportado num ambiente usual de simulação que, em geral, são somente capazes de rodar um modelo já determinado.

Na abordagem tradicional, muito do trabalho (senão todo) de modelamento é esforço humano. Resultados da simulação rodam, são avaliados e em casos de não-aceitação novos dados de entrada e parâmetros do modelo são fornecidos para execuções adicionais. Este processo é repetido até que o resultado desejado seja obtido, ou de outro modo, o modelo deve ser redefinido.

Estas tarefas podem ser feitas em Prolog

automaticamente. O modelador informa seu conhecimento sobre o sistema, especialmente a descrição dos objetos, as regras para modificação dos dados de entrada, parâmetros e características do modelo. Desta forma, a classe de todos os modelos possíveis e aceitáveis é definida.

Prolog possui uma extensão para a simulação discreta fornecida pelo T-Prolog.

Beb-Arieh [20] utiliza Prolog para construir um controlador para um SFM. O controlador utiliza uma base de conhecimentos para tomar decisões e estas decisões são, então, avaliadas por simulação como forma de testar sua eficácia.

5.3.10. Smalltalk

A tradição das linguagens algorítmicas procedurais, que teve início com o Fortran e ainda exerce influência nas principais linguagens em utilização atualmente, levaram os programadores a pensar de forma restrita sobre sistemas de software, ou seja, conjuntos de programadores independentes, partilhando apenas sub-rotinas e arquivos de dados. As linguagens possuem um número reduzido de operadores e tipos de dados, o que torna a programação uma operação minuciosa e afastada da semântica da aplicação.

Smalltalk é a mais popular linguagem orientada para objetos em uso atualmente e teve o mérito de lançar diversas concepções inovadoras sobre a construção de softwares.

Comparada com as linguagens tradicionais, o programador Smalltalk necessita de uma quantidade muito maior de conceitos para implementar suas aplicações. Já as linguagens convencionais possuem um conjunto pequeno de comandos, que são apreendidos com facilidade, mas isso não significa facilidade em implementar aplicações. Uma linguagem 'orientada para objetos' contrasta com a linguagem tradicional que é 'orientada para programas e dados'.

A primeira linguagem a ser provida de estruturas para definir classes de objetos genéricos foi a linguagem Simula, uma extensão do Algol 60. As idéias da Simula serviram de base para o Smalltalk, que foi desenvolvida no centro de pesquisa da Xerox, na década de 70. Além das idéias da Simula, a linguagem incorporou um novo conceito devido a Alan Kay, um de seus idealizadores, o princípio de objetos ativos prontos a 'reagir' a 'mensagens' que

ativam 'comportamentos' específicos do objeto. Assim, na linguagem os objetos deixam de ser meros 'dados' manipulados por 'programas' e são encarados como 'processadores' individuais e independentes, que podem receber comandos em forma de 'mensagens'.

Smalltalk, assim como outras linguagens orientadas para objetos, tem sido utilizada em várias aplicações onde a ênfase está na simulação de sistemas. Segundo **Jonathan** [110] são aplicações tais como: automação de escritórios, animação gráfica, informática educativa, editores de texto e bancos de dados genéricos. Estas aplicações são substancialmente diferentes de linguagens convencionais algoritmicas, que são mais apropriadas a problemas de: busca, ordenação, otimização e resolução numérica de equações.

5.4. CONSIDERAÇÕES

Não se conhecem trabalhos comparando as linguagens acima com respeito a adequabilidade para codificar modelos de simulação, mesmo por que tal trabalho seria muito difícil ou praticamente impossível, uma vez que estas linguagens possuem diferentes orientações. Um trabalho mais restrito envolvendo a comparação, através de vários testes, das linguagens Ada, C, Fortran, Modula-2 e Pascal como linguagens de simulação, foi elaborado por **Thesen** e **Sun** [259], que concluíram: "enquanto nenhuma linguagem dominou em todas as categorias, nossos testes sugerem que Turbo Pascal deve ser a escolha para pequenos projetos em micro e Ada para grandes projetos".

Uma opinião sobre algumas linguagens é colocada por **Crookes** [164] que diz: "eu tenho escrito, ou feito contribuições para, sistemas de variados graus de utilidade, incluindo algumas entidades visíveis de suporte e trabalho interativo, em Assembler, Algol 60, Neliac, Pascal e C. Eu tenho também estudado o trabalho interno de outros sistemas. De todos estes, minha atual preferência é por C, com Pascal correndo em segundo. Eu espero que Pascal venha a ser susbstituído deste segundo lugar em minha afeição por Modula-2 em algum tempo. Se C será da mesma forma substituído é discutível mas não será por Ada se isto vier a ocorrer".

6. CONJUNTOS DE SUB-ROTINAS (PACOTES)

6.1. INTRODUÇÃO

Apesar das facilidades oferecidas pelas linguagens de simulação e da disponibilidade destas linguagens envolvendo várias formas de programação, muitos analistas implementam modelos em linguagens de alto-nível de propósito geral [221]. A principal razão é que os modeladores estão normalmente familiarizados com estas linguagens e, é claro, com conceitos de simulação, mas estão relutantes (de má-vontade) para investir tempo e esforço para aprender uma linguagem de simulação de propósito especial [52]. A maioria das linguagens de alto nível (vistas no capítulo 5) não fornecem facilidades (rotinas, procedimentos ou funções) voltadas para simulação, uma vez que elas não são projetadas especificamente com esta finalidade. Para superar estas deficiências é que os entusiastas destas linguagens foram criando, ao longo do tempo, bibliotecas de facilidades voltados para a simulação que são denominados conjuntos de sub-rotinas ou pacotes.

Quando uma LPG é selecionada, então, ela é freqüentemente acrescida pelo uso de bibliotecas de rotinas pré-preparadas, ou escritas em casa, ou obtidas de um fornecedor externo. Em virtude do baixo custo de compiladores atualmente disponíveis, ligado à evidente aversão de muitos usuários de computadores a expectativa de aprender ainda um outro conjunto de regras sintáticas, a decisão em favor de tal LPG estendida é, freqüentemente, facilitada. Pacotes para melhorar as linguagens comuns estão diponíveis em várias fontes e muitas empresas têm suas próprias rotinas [164].

Em simulação, o programa executivo, que é a rotina de controle da simulação e responsável pela atualização da variável relógio (*clock*) que controla o tempo, é preparado independentemente de qualquer modelo particular e recorre as rotinas especificando o modelo. Assim, de certo modo, o executivo pode ser pensado como um programa 'principal' e o modelo como uma biblioteca suporte.

Apesar dos diferentes pontos de vistas de quais seriam as funções de um pacote de simulação, existe quase um consenso

quanto à seguinte lista de facilidades [39]:

1. Um mecanismo de controle do tempo para manter o relógio da simulação e a lista dos eventos a ocorrer, iniciar e terminar atividades e manusear a lógica do pacote.
2. Uma base de dados ou estrutura de arquivos para colocar os dados sobre os elementos do modelo em uma forma padronizada.
3. Um método para definir as condições inicais no modelo. Deve ser possível especificar onde cada entidade está, qual o conteúdo das filas e que atividades estão sendo executadas.
4. Geração de números aleatórios e amostras das principais distribuições.
5. Registrar observações, analisar resultados e imprimir relatórios.
6. Exibir histogramas e outras formas de resultados gráficos.
7. Diagnosticar e checar erros. O pacote deve ser capaz de detectar condições ilógicas ou impossíveis e fornecer mensagens apropriadas.
8. Exibir na tela o estado do modelo em qualquer ponto do tempo. Uma exibição animada do sistema sendo modelado fornece uma forma fácil para checar sua operacionalidade e apresentar seus resultados a quem interessar.

6.2. PACOTES DE BASE FORTRAN

6.2.1. Gasp

GASP é uma biblioteca de mais de 30 sub-rotinas e funções em Fortran que fornecem numerosas facilidades incluindo uma rotina de avanço de tempo (chamada Gasp), uma rotina para manejar a lista de eventos (isto é, adicionar e retirar eventos da lista de eventos a ocorrer), rotinas para adicionar e remover entidades de conjuntos, para coletar estatísticas, gerar variáveis aleatórias e um gerador padrão de relatórios.

Inicialmente, foi um pequeno conjunto de procedimentos para ajudar na análise de dados, programação, filas, números aleatórios e análise estatística. Com o uso, o GASP foi sendo continuamente expandido e hoje possui capacidades expressivas.

O pacote fornece uma estrutura organizacional que permite que o modelo seja descrito em termos de um modelamento

discreto, contínuo ou em uma combinação dos dois. No caso de modelamento discreto a abordagem é a do evento. O modelo contínuo pode ser descrito por equações diferenciais.

O programador deve estar familiarizado com a organização interna do pacote que requer habilidade para programar em Fortran, pois deve providenciar um programa principal, uma rotina de inicialização, rotinas de eventos, se desejado um gerador de relatórios, e mais uma rotina denominada *evnts*. O programa principal deve ter uma declaração denominada *call gasp* para iniciar a simulação. A sub-rotina *gasp* determina os eventos iminentes e chama a subrotina *evnts* que foi escrita pelo usuário com um índice, chamado *next*, em GASP IV. Este índice (*next*) indica que eventos estão para ocorrer, ou seja, para serem chamados por *evnts* [14].

GASP II foi a primeira publicação bem documentada do pacote, seguido por GASP IV que forneceu simulação mista discreta/contínua. GASP V expandiu as capacidades contínuas do GASP IV. Uma orientação adicional para o processo é dada por GASP VI.

O pacote é popular nas universidades americanas em virtude de seu preço baixo e sua base Fortran, mas tem sido substituído pelas linguagens Slam e Siman, que possuem algumas de suas características [39].

6.2.2. Simon

Simon é uma biblioteca de rotinas de simulação, implementada originalmente em Algol, mas desenvolvida em Fortran desde 1966. As rotinas básicas estão em sua sexta revisão. Elas fornecem facilidades para manuseio de filas, para programação de atividades e identificação de eventos. A estrutura 'preferida' é a das 3 fases, mas é também possível implementar uma abordagem orientada a eventos. As rotinas incluem ferramentas de redução de variância e estimação por intervalos.

O pacote estendido, para incorporar rotinas gráficas para animação do modelo, é denominado **Simong** [134]. O pacote é suportado pelo programa gerador **Draft** (ver capítulo nove).

6.2.3. Sdl

SDL (*Simulation Data Language*) é um conjunto de rotinas Fortran orientada que tem sido utilizada com **Q-Gert** e **Slam.** Ela permite ao usuário manipular dados, calcular estatísticas e fazer gráficos e histogramas de informações selecionadas.

O conjunto de rotinas tem sido interfaceado com pacotes de planilhas eletrônicas (como o Lotus 1-2-3) que são úteis na prepraração de saídas selecionadas, mas não servem as amplas funções de uma ferramenta de simulação específica [163].

6.3. PACOTES DE BASE PASCAL

6.3.1. Passim

Passim (*PAScal-based SIMulation*) é um conjunto de procedimentos em Pascal utilizando a abordagem de modelamento orientada a eventos. O pacote força o modelador a adotar um ponto de vista evento orientado. O modelador deve definir a espécie de evento que ocorre no sistema e especificar como estes eventos mudam o estado do sistema. Passim é escrito em Pascal ISO como forma de garantir a portabilidade.

Passim inclui um tipo dinâmico de entidade denominado de **transação** e três outros tipos de entidades que são a intalação (*facilitie*), o armazém (*storage*) e a cadeia (*chain*). As entidades instalação e armazém representam unidades e equipamentos de múltiplas capacidades, respectivamente. A cadeia representa listas de transações temporariamente inativas.

Passim fornece ao modelador um conjunto de procedimentos básicos, denominados 'procedimentos blocos', que representam eventos. A execução de um procedimento bloco significa a ocorrência de um evento e a modificação dos valores dos atributos do sistema de entidades.

Passim fornece também os procedimentos *on-chain* e *off-chain* para auxiliar o programador na manipulação de listas e que podem ser combinados para transferir uma transação de uma cadeia para outra. Os procedimentos *getspace* e *freespace* são empregados para minimizar necessidades de armazenagem. *Getspace* obtém espaço, dinamicamente, para transações e *freespace* libera o espaço quando uma transação termina. Outros procedimentos inicializam entidades, mostram estatísticas acumuladas e fornecem

as distribuições de probabilidades básicas.

Passim é um conjunto compacto de procedimentos Pascal, padrão que visa facilitar a implementação de modelos discretos utilizando a abordagem evento orientada. Nenhuma linguagem de simulação ou pacote garantem um modelo bem projetado e funcional. Isto é muito dependente da experiência e habilidade (prática) do modelador. Para utilizar Passim o modelador deve ser um programador Pascal competente, entendendo os conceitos básicos de modelamento e os procedimentos de programação Passim [221].

6.3.2. Simpas

Simpas é um pacote de simulação discreta utilizando a abordagem de modelagem orientada a eventos semelhante a linguagem Simscript. Comandos como: *schedule, cancel, destroy* e *reschedule* são utilizados para manipular informações sobre eventos. Comandos *insert* e *remove* são utilizados para manipular entidades em filas e as rotinas eventos são simplesmente procedimentos Pascal com a palavra chave '*event*' susbstituída por '*procedure*'.

O tradutor da linguagem é implementado como um pré-processador que aceita como entrada um programa Simpas e produz como saída um programa Pascal. Este programa Pascal deve ser compilado no computador desejado utilizando um compilador Pascal.

As características do Simpas levaram-no a ser utilizado como base para sistemas distribuídos (múltiplos processadores). Pesquisadores têm centrado esforços em torno de duas abordagens distintas para o projeto e implementação de sistemas de simulação discreta distribuído. A diferença reside em que tipo de tarefa é distribuída para os processadores individuais.

A abordagem mais comum é aquela em que funções do modelo (eventos) são distribuídos entre os processadores. A abordagem alternativa distribui funções de suporte (como geração de números aleatórios) para o processador disponível [232].

Uma diferença filosófica básica separa os dois métodos. Na simulação discreta distribuída via funções do modelo, os projetistas procuraram evitar bloqueios totais do sistema em situações onde o conjunto de eventos é maior que o número de processadores disponíveis com o objetivo de maximizar o paralelismo potencial do modelo. Aumenta-se a velocidade com

sacrifício da facilidade de programação. Na simulação discreta distribuída via função suporte, os projetistas escolheram uma abordagem que facilita a programação, sacrificando a utilização plena do paralelismo em favor de uma arquitetura do sistema 'transparente' para o usuário.

A implementação e execução de um sistema protótipo baseado no Simpas mostra que a abordagem distribuída para simulação discreta é uma alternativa viável para a simulação tradicional seqüencial. O usuário tem a vantagem de programar em uma linguagem que lhe é familiar, enquanto usufrui dos benefícios proporcionados pelo processamento paralelo [232].

6.3.3. Inter-Sim

É um pacote de simulação visual interativo evento orientado, escrito em UCSD Pascal. O pacote foi originalmente desenvolvido para uso em um PC sob DOS sendo posteriormente estendido para minicomputadores e é utilizado para modelar redes de comunicação e *layouts* de produção [191].

O nível de complexidade de modelos que podem ser diretamente construídos em Inter-Sim é comparativamente baixo. Desta forma são necessárias extensões. Isto é alcançado por reprogramação de rotinas em Pascal e religando com o Inter-Sim. A execução do modelo pode ser suspensa, permancecendo neste tempo e estado até ser novamente executada.

A facilidade com que um modelo pode ser desenvolvido é significativa. Alguns usuários vêem Inter-Sim como uma ferramenta para um protótipo rápido do modelo, sendo, então, convertido para uma linguagem geral ou de simulação [153].

6.3.4. Outros pacotes de base Pascal

São muitos os conjuntos de sub-rotinas que são extensões do Pascal voltadas para a simulação. Além dos relacionados acima, a literatura corrente inclui, entre outros: Pascal_Sim [64], SimTools [188], Simone [114], Pascal-Plus [226], eLSE [60], Turbo SIM [145], Pasmap [222], Passion [167] e Simul [165], sendo este último uma versão melhorada [165], em português, do eLSE. Conjuntos de sub-rotinas, não nomeadas, podem ser encontrados ainda em **Hác** [93] e **Kriz** e **Sandmayr** [118].

6.4. PACOTES COM BASE EM OUTRAS LINGUAGENS

6.4.1. Disc

Disc (*DIscrete event Simulation in C*) fornece uma estrutura para a criação e execução de modelos eventos discretos similares às características de programação de eventos de linguagens populares baseadas em Fortran tais como Slam, Siman ou ao pacote Gasp. O pacote fornece as rotinas necessárias, codificadas em C, para suportar simulação baseada em eventos. Em Disc as funções C requeridas são organizadas em 8 grupos (ou módulos) que são as tarefas de:

1. Controle executivo,
2. Acesso e armazenagem de informações,
3. Gerenciamento de memória dinâmica,
4. Geração de relatórios estatísticos e coleta de dados,
5. Monitoramento e acompanhamento do modelo,
6. Execução interativa,
7. Entradas e saídas do modelo e
8. Geração de variáveis aleatórias.

Pela utilização das várias funções incluídas nestes módulos, um sistema pode ser modelado pela escrita de rotinas eventos em C. Uma das mais atrativas características do pacote é a capacidade para executar a simulação de modo interativo.

As vantagens deste pacote estão associadas às vantagens da linguagem C, que está se tornando uma linguagem dominante no desenvolvimento de softwares para micros. A portabilidade do C assegura que programas escritos em uma plataforma computacional podem ser transportados para outra com poucas ou nenhuma modificação. Pode-se, por exemplo, desenvolver o programa em um micro e executá-lo em um mini ou mainframe [189].

6.4.2. Mex

Mex (*Modula-2 EXtended*) é uma extensão, como o nome indica, da linguagem de programação Modula-2. Foi desenvolvida nos laboratórios de Teleinformática da Escola Politécnica Federal de Lousane, Suíça.

À linguagem Modula-2 foram adicionadas rotinas para suportar múltiplos processos e inter-processos de comunicação e

sincronização. Em 1989, existiam implementações da MEX para estações de trabalho SUN e micros *Apple Macintosh* [151].

O pacote pode ser utilizado nos modos tempo real e tempo virtual, sendo que o último dos dois é utilizado para simular e testar protocolos. O modo tempo virtual emprega uma abordagem discreta evento orientada. Mex foi construída com o propósito de simular protocolos de comunicação de dados.

A principal vantagem do pacote é o suporte fornecido aos conceitos de engenharia de software. Mex fornece um ambiente que facilita o emprego de modularidade, tipos construídos, tipos abstratos, co-rotinas e compilação em separado [151].

Apesar de Mex ter sido originalmente projetada para suportar o desenvolvimento de sistemas distribuídos, e mais especificamente protocolos de comunicações de dados, o forte suporte aos conceitos de engenharia de software presentes na linguagem Modula-2 fazem de Mex uma linguagem de simulação geral e poderosa. Saliente-se que protocolos de comunicação são um dos principais problemas de um ambiente computacional industrial e particularmente dos sistemas flexíveis de manufatura.

6.4.3. Hpsim

Hpsim é uma coleção de procedimentos em Modula-2 que suporta o modelamento de sistemas utilizando as abordagens discretas evento orientada e processo orientada. Este pacote foi projetado para fornecer um código fonte alternativo para as linguagens Simula e Simscript ou o pacote Simpas. A abordagem do evento é semelhante à adotada por Simscript e Simpas e sua abordagem processo é de alguma forma similar aos da Simula e Simscript. Enquanto Simula e Simscript são compiladores e Simpas é um pré-processador, Hpsim é um código isolado que pode imediatamente ser incorporado a um modelo escrito em Modula-2.

O projeto dos módulos do Hpsim foi realizado utilizando-se os conceitos de dados abstratos e informação oculta. Cada módulo é implementado com o sentido de integralidade. Todas as operações necessárias para dar suporte à implementação de um tipo particular são fornecidas, enquanto os detalhes da implementação estão escondidos do modelador.

Modula-2 fornece uma *interface* natural para tipos, dados e operações do Hpsim através do uso de listas *import*. Utilizando

os padrões *export/import* um modelo de simulação legível e sustentável pode ser construído. As vantagens das técnicas de engenharia de software da Modula-2 são inteiramente repassadas ao pacote Hpsim uma vez que ele é implementado como um conjunto de módulos que são ligados com o código de simulação do modelador. Programação estruturada, desenvolvimento hierárquico, refinamento passo a passo, compilação separada e dados abstratos são métodos disponíveis em Modula-2 e que podem efetivamente ser utilizados no desenvolvimento de grandes modelos de simulação com Hpsim [195].

6.4.4. Apse

APSE (*Ada Programming Support Environment*) é um conjunto de pacotes (sub-rotinas) em Ada com o propósito de fornecer facilidades para simulação, incluindo as abordagens evento e processo. O conjunto é constituído de pacotes para manejar filas, coletar estatísticas, gerar números aleatórios e controlar a simulação. Apresenta, ainda, um pacote relógio. Destes pacotes o único que não é comum às duas abordagens é o pacote controle, cuja função é gerenciar a seqüência dos eventos e manter o relógio e pode ter a abordagem evento ou processo. O pacote 'relógio' é implementado como uma estrutura separada para facilitar seu acesso por outros pacotes. O pacote 'fila' gerencia a fila das entidades temporárias da simulação. O pacote 'números aleatórios' é (nesta versão) ainda bastante incompleto, apresentando apenas geração da uniforme e da exponencial. O pacote 'estatísticas' pode acumular dados, calcular médias, variâncias e imprimir resumos [78].

6.4.5. Simpl/I

SIMPL/I é uma extensão da linguagem PL/I para utilização em simulação e que, na versão 4.0, inclui programação objeto orientada e animação. Possui um ambiente de amigável incluindo assistência a sintaxe *on-line* e depurador integrado. O tempo de execução e tamanho do código do modelo foram reduzidos, em mais de 50%, em relação a versão 3.0.

6.4.6. T-Prolog

Uma extensão orientada para a simulação discreta,

derivada do Prolog, é o T-Prolog. As principais características do T-Prolog são:

- Adota abordagem de modelagem processo base.
- Oferece um mecanismo embutido que permite retroceder no tempo em caso de paralizações ou situações incorrigíveis que surjam durante a execução do processo.
- Pode mudar a estrutura original do modelo de forma automática com base em sofisticadas pré-condições [3].

6.5. CONSIDERAÇÕES

Para se valer destas coleções de rotinas o usuário deve escrever a estrutura do programa na linguagem base necessitando, deste modo, fluência considerável na linguagem. Isto pode ser vantajoso ou não. Se por um lado o analista não precisa aprender uma nova linguagem, uma vez que deve conhecer a linguagem base, por outro lado dificilmente tais pacotes fornecem uma estrutura global de simulação como as fornecidas pelas linguagens de simulação.

Existe ainda a dificuldade cultural, ou seja, um confronto de estilos de programação entre o que o programador deve construir e as rotinas já prontas. A adaptação nem sempre é muito fácil e as mensagens de erro geradas pelo pacote se referirão provavelmente às declarações da linguagem base e como tal podem ser difíceis de serem entendidas. Para evitar este problema pode-se, ao invés de comprar uma biblioteca comercialmente diponível, desenvolvê-la para uso próprio. Afora o eventual investimento de tempo inicial, a vantagem poderá ser o conhecimento preciso do funcionamento das rotinas, bem como o fato delas serem desenvolvidas sob medida para os problemas que irão resolver [163].

CAPÍTULO VII

7. LINGUAGENS DE SIMULAÇÃO

7.1. INTRODUÇÃO

A ampla aceitação da simulação como instrumento de análise levou ao desenvolvimento de um grande número de linguagens projetadas especificamente com este propósito. Estas linguagens incorporam as facilidades que nas linguagens de propósito geral só podem ser obtidas com a adição dos conjuntos de sub-rotinas específicas fornecendo, desta forma, um sistema integrado mais produtivo para a codificação de um modelo de simulação. O propósito deste capítulo é analisar estas linguagens que para facilitar este objetivo são agrupadas em linguagens: declarativas, de diagramas de blocos, de redes de filas e objeto orientadas, de acordo com a concepção de projeto.

A maioria das linguagens de simulação, atualmente em uso, possuem versões para *mainframe*, *mini* ou *micro* computador. Todas as linguagens incluem uma sintaxe disponível para simulação, um executivo e uma livraria de rotinas úteis.

Pode-se salientar as seguintes vantagens na utilização de linguagens de propósito especial ao se codificar um modelo de simulação [193]:

- Redução do trabalho de programação.
- Orientação na articulação de conceitos e formulação do modelo.
- Auxílio na comunicação e documentação do estudo.
- Flexibilidade no refinamento ou revisão do modelo.
- Fornecimento de funções usuais de suporte necessárias em qualquer simulação.

7.2. LINGUAGENS DECLARATIVAS

Linguagens declarativas, procedurais ou imperativas são as linguagens orientadas a programas e dados em oposição a orientação ao objeto das modernas linguagens objeto orientadas. Neste tipo de linguagem o programador deve especificar exatamente o que o computador deve fazer e como ele deve fazer, contrastando

com as linguagens lógicas (como o Prolog) onde o programador fornece regras e uma base de dados e o computador, então, com base nestas regras e na base de dados, tenta resolver o problema. São as primeiras linguagens de simulação, cuja orientação é em direção a problemas específicos de simulação.

7.2.1. Simula

Simula (*SIMUlation LAnguage*) é uma linguagem projetada para facilitar a descrição formal de *layouts* e regras de operação de sistemas discretos. Foi desenvolvida no centro de computação Norvegiam de Oslo por **Dahle Nygaard** em 1966, especificamente para a divisão Univac da Sperry Rand Corporation e é uma verdadeira extensão do Algol 60, isto é, contém o Algol 60 como um subconjunto [172]. A linguagem original implementou a abordagem do processo. A linguagem possui um grupo ativo de usuários denominado ASU (*Association of Simula Users*).

Simula tem suas origens do Algol, mas contém muitas avanços na estrutura **classe** (*class*). Uma declaração de 'classe' pode ser prefixada por outra 'classe' que compartilha propriedades comuns. Simula utiliza o termo **objeto** para descrever entidades. Desta forma, um objeto pode ser membro de uma classe, com atributos e uma história de vida. Esta história de vida, ou processo, é a seqüência de atividades ou ações em que objetos desta classe estão engajados. Os objetos devem ser identificados por uma declaração de tipo antes de serem utilizados. Uma das características do Simula é a sua capacidade para criar, destruir e modificar processos existentes ou novos criados por coleções de declarações Simula de controle.

Simula cria um arquivo de dados comum que é acessível por todos os processos. Simula também lida com atividades que podem ser criadas ou destruídas através de grupos estruturados de sentenças ou comandos. Uma transação pode ser criada ou destruída por processos, no entanto, os processos devem ser construídos pelo usuário para cada caso individual. A lógica do programa é controlada por uma rotina '*master clock*', e a linguagem fornece lógica pré-programada para conectar todos os componentes do modelo. Fornece, também, gerador de números aleatórios, vários esquemas de geração de variáveis aleatórias, saídas de formato fixado e instrumentos para a checagem de erros. Também permite ao

usuário programar capacidades especiais na linguagem se assim o desejar.

Segundo o criador da linguagem, ela é uma ferramenta apropriada para analisar redes neurais, sistemas de comunicações, de produção, administrativos e sociais, bem como fluxos de tráfego.

O pré-processador Sinon 75 de **Hills** e **Birtwistle**, inclui uma estrutura do modelo mais próxima do diagrama ciclo da entidade (DCE). Fornece, também, tipos de recursos pré-definidos para coleta automática de dados e geração de relatórios, programação automática e acompanhamento, total ou seletivo, de eventos.

Estas idéias foram adicionalmente desenvolvidas por **Birtwhistle** em 1979 [163] no pacote **Demos** (*Discret Event Modelling on Simula*). Este pacote foi construído para auxiliar iniciantes a desenvolverem programas na linguagem, forçando-os a utilizar uma estrutura simplificada.

7.2.2. Simscript

A linguagem Simscript (SIMulation deSCRIPtor) foi desenvolvida em 1964 por **Markowitz** e outros na Rand Corporation como um pré-processador que produzia sentenças em Fortran. Posteriormente, um compilador foi produzido e após alguns proprietários, foi vendida para a empresa **Caci**, que a comercializou como Simscript II.5, agora com uma implementação padrão [113].

A **Caci** está sediada nos EUA, como estão muitos usários do Simscript. Em 1980 existiam 400 instalações, 250 universidades usuárias e sete implementações da linguagem. Desde então uma versão para PC foi introduzida e também uma versão UNIX para uso em rede de computadores. Simscript II.5 possui um grande número de usuários dentro da área militar e todos os usuários são bem suportados por documentação e outras informações. A **Caci** produz uma revista especializada chamada Simsnips, que é um serviço útil de resumo de artigos em simulação e de análise de livros e conferências sobre o assunto [175].

Simscript é uma linguagem de simulação completa. Embora tenha sido originalmente projetada para análise de simulação, ela pode ser usada como uma linguagem de programação de propósito

geral. Ela requer um compilador especial, e está disponível em plataformas computacionais que vão de mainframes, minis e superminis até PCs. Neste último caso, um ambiente interativo conhecido como Simlab é fornecido como uma *shell* MSDOS [163]. PC Simlab controla todos os aspectos da simulação durante o desenvolvimento do programa, bem como durante a execução. Comandos gráficos e interativos estão disponíveis para facilitar as tarefas de desenvolvimento e experimentação.

Um modelo de simulação em Simscript consiste de uma introdução, um programa principal e subprogramas eventos. A introdução não é parte do programa executável e é utilizada para definir os elementos do modelo. Uma das principais funções da introdução é a de definir a estrutura estática do modelo, fixando nomes para entidades permanentes e temporárias, atributos e o conjunto de relações. Para cada evento o nome e os atributos correspondentes são definidos na introdução, bem como, definições de tipos de variáveis e conjuntos. As variáveis que coletam estatísticas são, também, definidas na introdução.

O programa principal é usado para inicializar variáveis, programar a ocorrência inicial de eventos e iniciar a simulação. Os subprogramas eventos são usados para definir a lógica associada com o processamento de cada evento no modelo. A chamada para o subprograma evento é programada pelo usuário, mas executada pelo programa de controle da linguagem.

Uma vez descrita a estrutura estática, na introdução, a próxima etapa é codificar o programa principal e os subprogramas eventos. Eles são codificados por declarações de propósito geral da linguagem unida com declarações especiais para criar e destruir entidades, manipular entidades em filas, obter amostras e programar eventos.

A linguagem é estruturada em níveis de forma a facilitar o seu aprendizado. Estes níveis são:

1. Estruturas simples projetadas para introduzir os conceitos aos não-programadores.
2. Declarações que são comparáveis em poder ao Fortran.
3. Declarações que são comparáveis em poder ao Algol ou a PL/I.
4. Sentenças que fornecem uma estrutura para modelagem, introduzindo os conceitos de entidades, atributos e conjuntos.
5. Sentenças para avanço do tempo, processamento de eventos,

geração de amostras e para coleta e análise dos resultados.

Os níveis de 1 a 3 cobrem características que podem ser encontradas na maioria das linguagens de alto-nível, com ênfase considerável na remoção de restrições de tipo do Fortran. Os níveis de 4 e 5 introduzem os conceitos de modelamento necessários para simulação.

Uma das principais atrações do Simscript como uma linguagem de programação e simulação é a sua sintaxe de forma livre e inglês-semelhante. Os programas são fáceis de ler e tendem a ser auto-documentados. Outro ponto forte é a flexibilidade de tipos de comandos para a obtenção de saídas. São fornecidas funções para o cálculo de estatísticas usuais e geração de variáveis aleatórias. O *debugger* da linguagem é extensivo e facilitado por sua estrutura inglês-semelhante [229]. Uma aplicação desta linguagem para estudar um SFM é encontrada na referência [107].

7.2.3. Ecsl

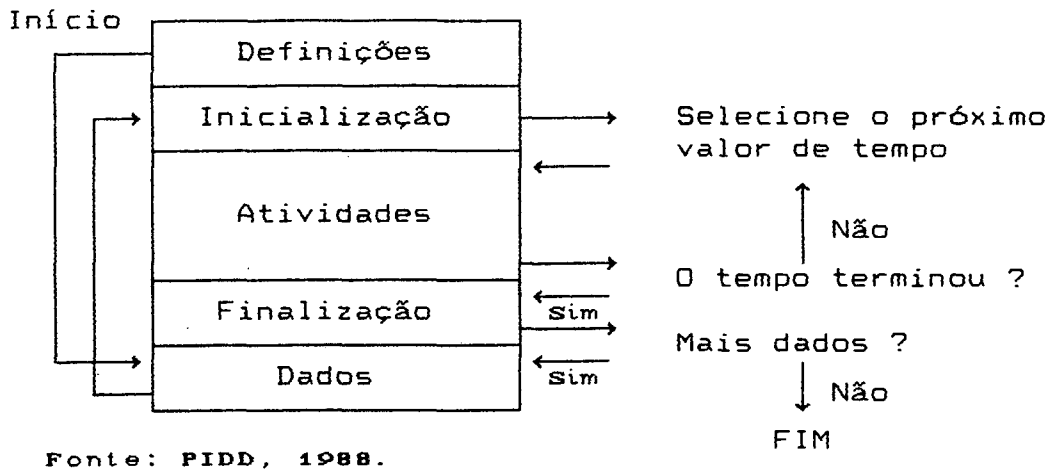
ECSL (*Extended Control and Simulation Language*) é uma linguagem popular inglesa que foi desenvolvida como uma extensão da CSL [54], que foi produzida pela Esso em 1960. As duas versões utilizam a abordagem exame de atividades. ECSL é interpretada, sendo o interpretador escrito em Fortran, podendo, desta forma, rodar em qualquer máquina que possua um compilador Fortran. No entanto, o usuário não precisa conhecer Fortran, nem o código traduzido em Fortran. A linguagem também pode rodar interativamente em sistemas multi-usuários ou em micros. É suportada pelo programa gerador **Caps** (ver capítulo nove) [163].

A linguagem apresenta várias características que são extensões significativas sobre o original. **Clementson** [54] afirma que implementou com êxito a **estrutura celular** [42] e que também incorporou **modelamento interativo visual** [19, 104].

Como outras linguagens de simulação ECSL fornece, entre outras, rotinas para amostragem e geração de números aleatórios. É inteiramente orientada a problemas de simulação e planejada para facilitar as tarefas de modelagem e programação. Os comandos são legíveis e poderosos, cada um correspondendo a vários comandos Fortran. Em virtude de ser interpretada, pode haver perda na eficiência da execução.

A linguagem ECSL assume que um programa de simulação é escrito com a estrutura de blocos da figura 7.1.

Figura 7.1 - Secções de um programa ECSL



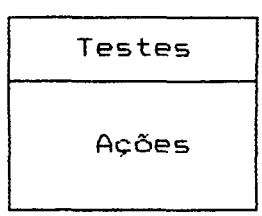
Fonte: PIDD, 1988.

Bloco 1. **Definições.** Estabelece as entidades e os conjuntos dos quais cada classe pode ser membro. Pode ser usado para estabelecer histogramas ou variáveis especiais necessárias.

Bloco 2. **Inicialização.** Fixa o estado inicial do sistema pelo estabelecimento dos estados das entidades específicas.

Bloco 3. **Atividades** (Encabeçado por comandos atividades). É o corpo principal do programa. Cada atividade é considerada como um bloco de código ECSL. A estrutura de uma atividade é vista na figura 7.2. Se as condições internas são tais que uma atividade não pode ocorrer, o controle passa para a próxima atividade.

Figura 7.2 - Estrutura de uma atividade ECSL.



Fonte: PIDD, 1988.

Bloco 4. **Finalização** (Iniciado por sentenças de finalização). Usado para produzir o relatório final do desempenho da simulação. Iniciado somente após completada a simulação.

Bloco 5. **Dados** (Iniciado por declarações de dados). Estabelece valores iniciais de algumas variáveis em conjunto com o bloco inicialização.

Quando um programa ECSL é executado, os seguintes processos ocorrem:

1. A secção 'definição' determina a memória necessária para o

programa e detalhes das entidades, conjuntos, atributos, etc.
2. O primeiro conjunto de dados é lido da secção 'dados'.
3. A secção de 'inicialização' estabelece as condições iniciais que são constantes para todos os conjuntos de dados da secção 'dados'.
4. Inicia-se a simulação propriamente dita e continua pelo tempo fixado. Durante esta fase existe um ciclo contínuo de exame de tempo e atividades.
5. A secção 'finalização' imprime as saídas.
6. Se existem mais dados, lê-se o próximo conjunto e volta-se ao passo três, caso contrário, pára.

Carrie et al [44] e **Mamalis et al** [133] utilizam esta linguagem para abordar problemas dos sistemas flexíveis de manufatura.

7.3. LINGUAGENS DE DIAGRAMAS DE BLOCOS

Para poder se valer de uma linguagem procedural ou um conjunto de sub-rotinas, o analista precisa, pelo menos, ser um programador razoável [163]. Linguagens de diagramas de blocos são uma tentativa para facilitar a tarefa de programação através da codificação do programa, somente em termos de blocos encontrados em um diagrama de fluxo. A programação é reduzida pela especificação dos blocos pelos quais uma entidade passa em sua trajetória pelo modelo. Esta mesma idéia é utilizada nos programas geradores (capítulo 9). A seguir, as principais linguagens deste tipo.

7.3.1. Gpss

GPSS (*General Purpose Systems Simulator*) é uma linguagem desenvolvida por **G. Gordon**, em meados de 1960 [87]. Originou-se de um trabalho desenvolvido nos laboratórios da Bell telefones dos EUA e assistido pela IBM. A versão original do GPSS foi escrita para auxiliar o desenvolvimento de redes de comunicações e foi destinada a usuários não especializados em computação. Ela é baseada na idéia de que o fluxo das entidades do sistema pode ser modelado por um diagrama de blocos.

Um programa fonte GPSS consiste de uma lista de blocos

através dos quais cada classe de entidades deve passar durante sua permanência no modelo. Cada bloco é uma linha distinta do programa, consistindo de um descritor do bloco (nome que identifica seu tipo ou ação) seguido de um ou mais parâmetros. Algumas versões do GPSS incluem um bloco *help*, através do qual o usuário pode especificar rotinas em Fortran ou PL/I.

Existem cerca de 40 blocos padrão em GPSS. Cada bloco representa uma ação específica ou evento que pode ocorrer em um sistema típico. Os blocos são organizados em um diagrama que representa um processo. O grande número de blocos é uma característica desagradável do GPSS e diminui muito as vantagens da representação diagramática simples, devido ao esforço de lembrar precisamente que ação no modelo é empreendida por qual bloco.

GPSS é provavelmente a linguagem de simulação discreta mais utilizada. As razões para esta popularidade são a facilidade de aprendizagem, especialmente por não programadores e o tempo relativamente curto para se construir modelos complexos [14].

A linguagem, no entanto, apresenta deficiências que podem tornar a modelagem de certos sistemas especialmente incômoda [14], como por exemplo:

1. O relógio da simulação pode assumir somente valores inteiros resultando que qualquer cálculo é sempre truncado (arredondado para menos).
2. Cálculos numéricos complexos e/ou ações lógicas podem ser extremamente difíceis, senão impossíveis, pois a linguagem não pode manusear logarítmos, senos, cossenos, exponenciais, valor absoluto, máximo e outras funções matemáticas comuns. Teoricamente a linguagem pode aproximar estas funções através de uma função linear ou pelo uso do bloco *help* que permite chamar rotinas em Fortran e PL/I.
3. Não possui geração de variáveis aleatórias embutida.
4. Os oitos fluxos de números aleatórios produzem a mesma seqüência, a menos que o programador explicitamente defina sementes diferentes para cada fluxo. Deve-se notar que GPSS/H eliminou a maioria das deficiências.

Três versões principais do GPSS são, atualmente, utilizadas: (i) GPSS/V, (ii) GPSS/H e (iii) GPSS PC. GPSS/V é um descendente do GPSS/GPSS III e é reconhecido como a *'de facto'*

versão padrão do GPSS. GPSS/H é um compilador e um pacote de suporte *'run-time'* para GPSS e é quase 100% compatível com as versões anteriores. GPSS PC é a implementação em micro do GPSS V com algumas extensões.

Como o GPSS foi uma das primeiras linguagens de simulação, possui um grande suporte bibliográfico e é talvez a única linguagem de simulação com uma referência, **Strack** [206], em Português. As aplicações aos sistemas flexíveis de manufatura, referências [11], [50], [86], [95], [190], [204], [224] e [228], entre outras, são também numerosas.

7.3.2. Siman

Siman (*SIMulation ANalysis*), desenvolvida em 1982, por **Dennis Pedgen**, da Universidade Estadual da Pensilvânia, marca a transição entre a idéia de uma linguagem de simulação como ferramenta para a representação simples de um modelo e a linguagem de simulação como uma técnica de resolver problemas [159].

Siman emprega diagramas de blocos como meio primário de modelar sistemas discretos, podendo modelar, também, sistemas contínuos ou mistos. Os blocos, cujas formas indicam suas funções, são montados com setas indicando a direção do fluxo da entidade. O diagrama de fluxo é construído como uma seqüência de blocos cujas formas indicam sua função geral. Um programa suporte especializado, denominado *blocks*, é utilizado para facilitar as entradas em forma de blocos, que são 10 ao todo, bem menos do que os 40 do Gpss. Os 10 tipos básicos de blocos formam 40 diferentes funções elementares. Os blocos são multi-funcionais, com exceção de três, e fornecem facilidades para pesquisa de arquivos, remoção de transações e cálculos aritméticos. A linguagem, não fornece um editor de comandos, desta forma, se o usuário não quiser se utilizar da entrada por blocos, será obrigado a integrar um editor ao software.

Siman é projetado em torno de uma estrutura de modelamento lógica em que os problemas de simulação são segmentados em uma componente 'modelo' e uma componente 'experimento'. Esta estrutura é baseada nos conceitos teóricos sobre sistemas desenvolvidos por **Ziegler** [235].

O modelo descreve os elementos físicos do sistema (máquinas, trabalhadores, pontos de armazenagem, transportadores,

etc.) e suas inter-relações lógicas (fluxo de componentes, de informações, etc.).

O experimento especifica as condições sob as quais o modelo é executado, incluindo elementos tais como: condições iniciais, disponibilidade de recursos, tipos de estatísticas coletadas e tamanho da corrida de simulação. A estrutura experimento também inclui as especificações do analista para tarefas tais como a programação da disponibilidade de recursos, as rotas das entidades, etc. Em virtude de condições experimentais serem especificadas externamente à descrição do modelo, elas são facilmente modificadas sem a necessidade de alterações nas definições básicas do modelo.

Uma vez que um 'modelo' e um 'experimento' tenham sido definidos, eles são ligados e executados pelo Siman para gerar as respostas simuladas do sistema. Uma vez executada a simulação, Siman armazena automaticamente as respostas especificadas no experimento. O processador de saídas Siman, denominado *playback*, pode ser usado para gerar gráficos, tabelas, histogramas, correlogramas e intervalos de confiança dos dados armazenados. Saídas animadas são obtidas através de um programa suporte chamado Cinema.

A linguagem inclui características especiais que facilitam a simulação de sistemas de manufatura e movimentação de materiais. Estas características incluem procedimentos para simular esteiras, robôs, AS/RS e células de manufatura. Estações são uma ferramenta especializada do Siman para representar equipamentos de movimentação de materiais motivado por pesquisas que mostram que 50% a 70% das atividades dentro de uma área de produção são causada por movimentos. A linguagem inclui blocos que podem representar os mecanismos comuns de transporte. Desta forma embute-se construções de manufatura de propósito especial dentro da estrutura de uma linguagem geral. Estações são definidas tal que movimentos de transporte por AGVs ou condução em esteiras podem ser feitos de estação para estação. O tempo de viagem entre estações é calculado como uma função da distância e velocidade do mecanismo, ambas definidas pelo usuário [159].

A seguir, um resumo das principais características do Siman [76]:

- Compatibilidade entre versões para micro, mini e mainframe de

forma a permitir a transferência entre sistemas computacionais sem modificações;

- Capacidade gráfica interativa para construir, definir e apresentar saídas do modelo;
- Um *debugger* interativo para monitoramento e controle da execução da simulação;
- Um sistema denominado Cinema, que gera uma animação, em tempo real, gráfica-colorida e de alta-resolução do sistema modelado;
- Uma estrutura modular que encoraja integração com outras tecnologias de análise e interfaces inteligentes.

As referências [127] e [168] são aplicações de simulação, em Siman, aos sistemas flexíveis de manufatura.

7.4. LINGUAGENS DE REDES DE FILAS (*QUEUING NETWORK*)

Sistemas flexíveis de manufatura, bem como outros sistemas, podem ser representados como modelos de redes. Uma rede é uma representação gráfica que possui alguma analogia com o diagrama ciclo da entidade (DCE). O conceito e a terminologia das redes está ligada a teoria dos grafos. Um grafo consiste de um conjunto de pontos de junção denominados **nós** (ou também **pontos** ou **vértices**), com certos pares de nós unidos por linhas denominados **ramos** (ou também **arcos, ligações** ou **arestas**). Uma **rede** é um grafo com um fluxo de algum tipo em seus ramos [99, 223]. Se ao conceito de rede associarmos o de fila teremos o conceito de **rede de filas**, isto é, rede de instalações de serviços, onde os clientes devem receber serviços de uma ou todas as instalações.

Uma terminologia associada a um modelo de redes de filas é reconhecer dois tipos de entidades: os consumidores (usuários dos serviços) e os produtores (fontes dos serviços). Por exemplo, quando uma peça é transportada em uma esteira para visitar várias máquinas para processamento, as peças são os consumidores, "consomem" espaço na esteira, e a esteira é o produtor, "produzindo" quantidades fixadas de espaço. Entretanto, a peça também consumirá outros serviços, como os produzidos pelas máquinas que visita.

Produtores podem ser tratados como os nós da rede e os consumidores como os fluxos passando pelos nós e que causam realizações de atividades que são os ramos da rede.

Várias linguagens utilizam conceitos de redes, como seu blocos de estruturas, normalmente pela definição de símbolos específicos para nós e arcos, sendo que a mais antiga e que introduziu este conceito em simulação é a Q-GERT.

7.4.1. Q-Gert

Q-GERT (*Queue - Graphical Evaluation and Review Technique*), é escrita em Fortran IV e fornece um sistema de entidades, denominadas transações, fluindo na rede de nó a nó através de arcos. Nós representam filas e decisões ou pontos de coleções de dados. Os arcos representam duração de atividades. A linguagem admite diferentes tipos de nós, como forma de permitir a modelagem de situações complexas e projetos de sistemas administrativos. Fornece coleta automática de estatísticas sobre muitas entidades padrões do sistema e sobre uma ou várias execuções da simulação. O relógio do Q-GERT é real e fornece funções para todas as variáveis aleatórias mais comuns [229].

A forma de se construir um programa em Q-GERT é semelhante a utilizada em GPSS. O modelador combina elementos em um modelo de rede que é uma representação gráfica do sistema de interesse. Este modelo de rede é então transcrito em registros de entrada para interpretação e processamento pelo programa de análise da linguagem.

A linguagem apresenta 10 tipos de nós. Este pequeno número é conseqüência da filosofia de acrescentar ou combinar funções nos nós, quando se tornam necessárias. Q-GERT também facilita o uso de programação Fortran, que pode ser inserida tanto em nós quanto em arcos.

Os atributos são utilizados para diferenciar transações. O valor de um atributo pode ser estabelecido em cada nó. A duração de atividades é determinada por uma distribuição e um conjunto de parâmetros. Arcos e nós têm vários atributos fixados. Por exemplo, os nós têm:

- Número do nó.
- Número inicial de transações na fila.
- Número máximo de transações permitidas na fila.
- Procedimentos para priorizar transações.

Arcos têm:

- O número da atividade.

- Probabilidade de iniciar a atividade.
- Tipo de distribuição para ramificações.
- Números de parâmetros fixados por ramificação.
- Número de servidores diferentes na ramificação.

Existem somente oito símbolos básicos e a linguagem é apresentada como uma lista parametrizada de nós ou ramos. Para dar poder a linguagem os campos dos parâmetros são longos. Alguns nós possuem até 20 parâmetros possíveis e outros, com repetições, podem passar dos 50 ítens.

Abdin e **Mohamed** [2] utilizam um programa de simulação em Q-Gert para determinar o número de *pallets*, velocidade de esteiras e regras operacionais de um SFM.

7.4.2. Slam

SLAM (*Simulation Language for Alternative Modeling*) é uma linguagem de simulação com base Fortran, introduzida originalmente em 1979. SLAM II, última versão do SLAM, suporta modelagem contínua e as abordagens de modelagem discretas evento base e a processo base ou, ainda, uma combinação das três.

Nós, em SLAM, são pontos de processamento que podem causar a chegada de uma entidade para esperar em uma fila, mudanças em valores de variáveis, coleta de estatísticas ou liberação de um recurso. Outros nós fornecem pontos de entrada ou saída da rede. Ramos (ou atividades) representam as rotas das entidades entre os nós. Roteiros podem ser determinísticos, probabilísticos ou baseados nos estados da variável. Atrasos de tempo ocorrem nos ramos representando tempos de processamento, de viagens ou de espera. O processador SLAM faz simultaneamente análise da rede e fornece estimativas do desempenho do sistema em relatórios padrões resumidos que podem ser complementados com saídas definidas pelo usuário [160].

A abordagem do processo é muito semelhante ao Q-GERT, empregando uma estrutura de rede composta de nós e ramos. Um conjunto de nós especializados e símbolos de ramos formam as componentes de modelagem da rede. Estes símbolos representam elementos do processo, tais como, equipamentos, áreas de armazenagem e pontos de decisão. A tarefa da modelagem consiste em incorporar estes símbolos em um diagrama de rede que representa o sistema. As entidades no sistema (representando, por exemplo:

peças e informações) fluem através do modelo de rede.

Na abordagem evento orientada o modelador define eventos como sub-rotinas Fortran, codifica o processamento lógico correspondente a cada tipo de evento em rotinas suportes separadas. O usuário escreverá uma sub-rotina para estabelecer as condições iniciais para a simulação e rotinas eventos para especificar as mudanças no estatus do sistema para cada evento.

Um modelo contínuo é codificado em SLAM pela especificação da equação algébrica, diferencial ou à diferenças que descreve o comportamento dinâmico do sistema. A linguagem automaticamente as integra para calcular os valores do estado das variáveis dentro de uma precisão estabelecida pelo modelador.

Sistemas mistos podem ser modelados através de uma abordagem evento e/ou processo. SLAM também incorpora características que correspondem a uma abordagem exame de atividades. Interfaces entre as abordagens são projetadas para facilitar simulação combinada.

Um aspecto importante do SLAM é que as diferentes abordagens podem ser combinadas dentro do mesmo modelo de simulação. Existem 6 interações específicas que podem ocorrer entre as abordagens de rede discreta/contínua da linguagem [229]:

1. Entidades no modelo de rede podem iniciar a ocorrência de eventos.
2. Eventos podem alterar o fluxo de entidades no modelo de rede.
3. Entidades no modelo de rede podem ocasionar mudanças instantâneas nos valores das variáveis de estado.
4. Variáveis de estado, alcançando o início dos valores prescritos, podem iniciar entidades no modelo de rede.
5. Eventos podem causar alterações instantâneas nos valores das variáveis de estado.
6. Variáveis de estado alcançando o início de valores prescritos podem iniciar eventos.

SLAM foi projetada e programada por **Dennis Pedgen**, com base no pacote GASP IV e na linguagem Q-GERT, sendo posteriormente apoiada pela empresa Pritsker e Associados. Ela é um superconjunto do Q-GERT e também linguagem base para o sistema TESS (ver secção 9.2.1). Ela tem sido continuamente revisada. MHEX apareceu em 1986 como uma extensão do SLAM II com estruturas para movimentação de materiais, especialmente para modelamento de AGVs [156].

Para rodar SLAM II em virtualmente qualquer mainframe, basta um compilador Fortran 66. Uma versão para PC foi desenvolvida, mantendo compatibilidade com a versão para mainframe [14].

SLAM parece ser tão ou mais popular que GPSS para aplicações em SFM. Estas aplicações incluem as referências [32], [46], [53], [55], [129] e [173].

7.4.3. Resq

RESQ (*Research Queuing Package*) é um produto da IBM. A linguagem fornece tanto abordagem analítica quanto de simulação de uma mesma definição do modelo. Com certas restrições o modelo pode ser resolvido analiticamente se, entretanto, as restrições não são viáveis, uma abordagem de simulação discreta poderá ser utilizada. A existência de uma estrutura subjacente ao modelo de simulação permite outros algoritmos para "resolver" o modelo e estimar alguns parâmetros de desempenho do sistema. Normalmente, estes parâmetros são um subconjunto restrito dentre os disponíveis em um experimento completo de simulação, por exemplo, o tamanho médio da fila, distribuição do tamanho da fila, ou tempo dado de espera. Isto contrasta com a informação completa disponível da trajetória do tamanho da fila que é gerada por uma simulação. Se a rede pode ser matematicamente analisada, o pacote RESQ possui uma estrutura para fazê-lo via o software de solução numérica QNET4.

RESQ fornece um conjunto de estruturas de modelagem específicas para problemas com recursos limitados, como por exemplo, sistemas computacionais, redes de comunicação e processos de manufatura. RESQ também tem um alto nível de suporte para análise, fornecendo ferramentas de simulação específicas para estimação de intervalos de confiança como os métodos regenerativo, espectral e replicações independentes [14].

Exemplos de aplicações da linguagem a problemas de manufatura são encontradas nas referências [51] e [150].

7.4.4. Resqme

RESQME (*RESearch Queuing package Modelling Environment*), é um sistema integrado de hardware e software orientado para a utilização de gráficos. Foi projetado para

facilitar o uso de modelos para propósitos de avaliação de desempenho. Gráficos são utilizados porque os projetistas acreditam que o analista não só pensa em termos de um diagrama de rede quando constrói seu sistema, mas também prefere ter as saídas em termos gráficos. O hardware consiste de um PC conectado a um mainframe com um vídeo de capacidade gráfica. O monitor gráfico usa uma interface IBM '*virtual display*' provida de um kit gráfico. O modelo é 'avaliado' no mainframe mas RESQ controla toda a interação entre a estação de trabalho e o computador principal.

As análises de saída são simples. O usuário aponta para um nó ou grupo de nós e seleciona 'um conteúdo específico' que produz uma lista de variáveis de saída disponíveis. Selecionando uma variável e pressionando '*plot*' gera-se automaticamente um gráfico com um intervalo de confiança. Para estimar os parâmetros de saída três técnicas de estimação por intervalo estão disponíveis para o usuário, que são a regenerativa, espectral e replicações independentes.

7.4.5. Outras linguagens de redes de filas

Saint (*Systems Analysis of Integrated Network of Tasks*) foi projetada para modelamento do desempenho de operadores humanos. É uma linguagem derivada do Q-GERT que foi desenvolvida pela USAF na Base Aérea de Wright Petterson. Uma versão para micro, denominada Micro Saint, foi lançada em 1985 [123].

Simnet (*SIMulation NETwork*) é uma linguagem de redes que utiliza somente quatro nós. Um nó 'fonte', para criar transações, um 'fila', para esperas, um 'processador' onde os serviços são executados e um 'auxiliar', introduzido para aumentar a flexibilidade de modelagem da linguagem. Cada nó é provido com informações para definir a maneira como a transação entra, reside e deixa o nó.

A linguagem foi projetada com três objetivos principais:

1. Aumentar a facilidade de uso pela eliminação de nós de propósito especial;
2. Obter flexibilidade sem a utilização de sub-rotinas ou funções (em Fortran);
3. Integrar os aspectos estatísticos do experimento da simulação diretamente na linguagem.

Simnet possui versões para micro, mini e mainframe. A

execução da linguagem pode ser interativa ou em *batch*. O modo interativo fornece informação instantânea sobre o sistema como o progresso da simulação através do tempo e é um auxiliar para a correção do modelo [210].

7.5. LINGUAGENS OBJETO ORIENTADAS

Os softwares anteriores são de linguagens orientadas a programas e dados. Desenvolvimentos recentes em software têm alterado esta ênfase e o interesse da simulação tem acompanhado. Linguagens objeto orientadas (OOP) concentram-se em objetos. A idéia de objeto é organizar e armazenar pedaços de informação relacionados a único conceito em um único local. Os pedaços de informação podem incluir fatos sobre o objeto, como ele se comporta sobre certos estímulos e com quem interage. OOP fornece uma maneira de elevar a representação do modelo para um nível mais alto de abstração do que seria conseguido com as linguagens procedurais tradicionais. Programas objeto orientados são escritos em termos de 'objetos' (também denominados de esquemas ou estruturas) ao invés de procedimentos. A filosofia da OOP é apropriada para sistemas que envolvem explicitamente a passagem do tempo e/ou mudanças de objetos no tempo. Esta filosofia pode ser resumida por:

- O usuário inicialmente cria ou define objetos que correspondem a objetos do mundo real.
- O comportamento dos objetos do modelo de simulação descrevem o comportamento dos objetos do mundo real e como estes objetos se comportarão em resposta a várias entradas.
- Objetos agem uns sobre os outros pela passagem de mensagens descrevendo tanto ações funcionais quanto relacionais. Ou seja, objetos possuem a habilidade de herdar características de outros objetos.

Desta forma, programação objeto orientada trata um programa como uma coleção de objetos que executam ações enviando ou recebendo mensagens. Em essência, o 'mundo' objeto orientado do ambiente de simulação consiste de pacotes de informação que fornecem regras comportamentais (objetos embutidos) e especificação de manipulação (mensagens embutidas). A abordagem objeto orientada é especialmente valiosa no sentido de que fornece

uma correspondência muito próxima entre objetos simulados e objetos do mundo real.

Na implementação de uma OOP, a definição de um objeto é, em geral, da seguinte forma:

objeto (<nome>, <propriedades>, <comportamentos>)

Os benefícios da programação objeto orientada encontram-se em duas simples, porém poderosas, características: encapsulamento e herança. Encapsulamento refere-se ao fato que tipos específicos de dados e as formas de manipulá-los podem ser colocados combinados em uma 'classe'. Herança significa que classes podem ser oganizadas em árvores (denominadas hierarquia hereditária ou rede semântica) de modo que novas classes possam herdar informações (fatos e métodos) de seus ancestrais. Estas características também tornam a escrita de programas em linguagens objeto orientadas altamente re-utilizável.

Por exemplo, a classe 'torno' representa um subconjunto específico de uma classe superior 'máquina' que é usada para tornear peças. Um 'torno vertical' é uma subclasse da classe 'torno'. Desta forma, pode ser descrito uma hierarquia complexa de objetos [192].

Embora no contexto da simulação a idéia de objeto não seja nova, uma vez que já era adotado pela linguagem Simula, a importância desta abordagem para a comunidade de programação, em geral, assegura o crescimento desta área. Para atestar a importância desta orientação, duas das principais linguagens de programação atualmente em uso desenvolveram pré-processadores para programação objeto: a linguagem Pascal através do Pascal Objeto e a C através do C++.

Antigamente, softwares objeto orientados requeriam significativo suporte de hardware e isto tornou lenta a aceitação desta técnica. Entretanto, novas implementações e micros mais poderosos aceleraram o crescimento desta técnica pelo crescimento da disponibilidade de ferramentas.

Hoje é possível interfacear uma linguagem descritiva como Prolog, voltada à inteligência artificial, com uma linguagem objeto orientado como Smalltalk. Pesquisadores têm, também, projetado ferramentas para fornecer uma descrição hieráquica do sistema para permitir entradas gráficas do sistema e também algoritmos de predição do desempenho e capacidades de diagnóstico

de bases de conhecimentos, possibilitando uma substancial redução nos custos de desenvolvimento de modelos.

Smalltalk é uma linguagem interpretativa e, naturalmente, a execução é lenta. No entanto, o conceito de código re-utilizável implica que uma vez que o código tenha sido escrito para descrever um objeto, digamos uma máquina, o objeto pode ser arquivado em uma biblioteca para uso posterior.

Um exemplo de um pacote objeto orientado na área de manufatura é o Simmek (Capítulo 8).

7.6. CONSIDERAÇÕES

Apesar das facilidades oferecidas pelas linguagens de simulação não se pode afirmar que sua utilização tenha superado a das linguagens de propósito geral para codificar programas de simulação, como comprova o levantamento realizado em **Thomas** e **DaCosta** [216], reforçado pela constatação de **Pidd** [163].

Comparações entre as várias linguagens de simulação são também difíceis de realizar em virtude das diferentes concepções e orientações destas linguagens. No entanto, um estudo deste tipo foi realizado comparando as linguagens Slam (rede de filas), Gpss/H (diagrama de blocos) e Simscript (declarativa) para simular três modelos de sistemas de manufatura sendo um pequeno, um médio e um grande. Cada programa do mesmo modelo foi escrito por um especialista na linguagem. A conclusão obtida por **Abed** [1] foi "a simulação do modelo de manufatura utilizada no experimento mostrou que Gpss/H é compilado e executado mais rápido e utiliza menos tempo de CPU e memória do que as outras duas linguagens. Simscript foi mais eficiente do que Slam para simular grandes modelos de manufatura por qualquer período de tempo e pequenos e médios por curtos períodos. Slam se mostrou melhor para executar modelos pequenos e médios por longos períodos".

Uma constatação tirada do exame da bibliografia, relatando a codificação de modelos de sistemas flexíveis de manufatura é que as linguagens mais utilizadas são Gpss e Slam, seguidas por Ecsl e Siman.

CAPÍTULO VIII

8. PACOTES ORIENTADOS POR DADOS (*DATA-DRIVE*)

8.1. INTRODUÇÃO

Estes pacotes envolvem os modelos genéricos de manufatura e os modelos genéricos de sistemas de manufatura, havendo sobreposição, pois muitos modelos genéricos de manufatura são também modelos genéricos de SFM.

Estes softwares não são linguagens, mas sistemas que praticamente não exigem programação. A idéia é que os detalhes do modelo particular devem entrar como 'dados' para o modelo genérico. Um modelo genérico é apropriado a uma classe particular de sistemas. Por exemplo, uma organização industrial pode utilizar tal modelo genérico precisando apenas de pequena reprogramação. Alternativamente, o modelo genérico pode ser geral o suficiente para permitir que as características da indústria sejam especificadas puramente com dados para o modelo.

O pacote **Hocus** (secção 8.2.10) é um exemplo de modelo genérico com uma larga variedade de aplicações. A principal vantagem de tais pacotes é a velocidade na qual aplicações particulares podem ser desenvolvidas, muitas vezes, por equipes relativamente inexperientes. Isto é, eles podem ser muito fáceis de se usar. Um modelo genérico bem-escrito pode ser utilizável por equipes que conhecem pouco ou nada de seu trabalho interno. Em tais casos, o usuário pode estar familiarizado com um sistema particular sendo simulado, mas não com os detalhes da simulação computacional. A principal desvantagem é uma conseqüencia da facilidade de uso. Existem situações em sistemas a serem simulados que se adaptam muito mal na estrutura imposta por modelos genéricos. Em alguns casos, esta dificuldade pode ser superada se o simulador possui a habilidade para recorrer a uma linguagem geral (C, PL/I, Fortran, etc.) para programar decisões lógicas complicadas. No entanto, em geral, este tipo de aplicação particular será melhor simulada valendo-se de outros softwares que não modelos genéricos.

A programação através de um pacote dirigido por dados

consiste, normalmente, em fornecer dados numéricos. Tais informações podem representar uma simples contagem de máquinas ou uma tabela de tempos de operação para cada processo no roteiro de um tipo particular de peça. A natureza destas informações é tal que, uma vez coletadas no sistema, será somente necessário colocá-las no formato apropriado para poder executar a simulação. Este conceito está muito próximo da simulação automatizada.

Uma estrutura típica destes recursos envolve um(a):

Editor e criador do modelo, que pega uma descrição interativa dos detalhes do sistema a ser simulado e produz um arquivo que pode ser utilizado posteriormente ou que serão dados para a própria simulação. O arquivo fornece uma descrição completa do sistema a ser simulado. Este criador/editor deve controlar o modelo conforme as regras exigidas pelo simulador.

Simulador, que inclui as estruturas normalmente encontradas em um programa de simulação discreta.

Manipulador de interações, que permite e controla a interação do usuário com a simulação quando ela é executada.

Biblioteca gráfica, para permitir a visualização do esquema lógico e exibir gráficos na tela quando a simulação estiver sendo executada.

Gerador de relatórios, para facilitar a produção de relatórios do desempenho do sistema sendo simulado.

8.2. MODELOS GENÉRICOS DE MANUFATURA

Os modelos mais simples deste tipo permitem ao usuário definir estações de trabalho, indicar a proporção de peças fluindo de um centro de trabalho para outro, o tempo de operação para cada centro de trabalho e a taxa de chegada das peças a serem processadas. Estatísticas da utilização de cada centro de trabalho e do tamanho das filas são, então, produzidas. Pode, também, ser possível interagir com o modelo para causar paralizações, gargalos ou outros efeitos e observar o tamanho das filas. Pacotes mais sofisticados permitem ao usuário especificar as regras de processamento para vários tipos de peças e distinguir entre vários tipos de recursos e operações. Exemplos deste tipo de pacote são o Modelmaster, Xcell, Simfactory e Witness, sendo este último, conforme **Carrie** [39], um dos mais desenvolvidos.

Um pacote orientado por dados voltado para a manufatura é denominado de **simulador**. Ele permite a simulação de classes específicas de sistemas de manufatura com pouca ou nenhuma programação. O sistema particular de interesse (no domínio do pacote) é tipicamente codificado através de menus ou gráficos sem a necessidade de programação. A principal vantagem é que o tempo de desenvolvimento do programa é, normalmente, bem menor do que aquele gasto utilizando outro tipo de software. Isto pode ser importante devido às restrições de tempo da maioria dos ambientes de manufatura. A principal desvantagem dos simuladores é que eles são limitados à modelagem daquelas configurações de manufatura permitidas por suas características padrões [124].

8.2.1. GSP

GSP (*General Simulation Package*) foi o primeiro pacote de simulação britânico. Na Inglaterra, o trabalho pioneiro em simulação foi feito na indústria do aço por **Tocher** e outros. Como resultado da experiência em modelar vários aspectos do trabalho com aço **Tocher** observou que muitos modelos tinham características comuns e, sobre isto, desenvolveu um modelo genérico que incluia muitas das características observadas.

Virtualmente, todo o desenvolvimento em simulação na Inglaterra foi conseqüência deste trabalho e foi feito por seus antigos colegas ou por aqueles que foram influenciados por eles [39].

8.2.2. Simfactory

Simfactory é um pacote de simulação voltado ao ensino de projeto de fábricas e análise de produção. O layout da fábrica e os parâmetros da produção são fornecidos através da interface Simfactory com o usuário, um dos dois sistemas componentes.

Como um sistema dirigido por dados, cada execução do modelo deve conter o conjunto completo de dados. Para uma célula com Robô estes arquivos contém:

Layout da célula,
Descrição do Produto,
Programação da Produção,
Especificação de relatórios.

Estes arquivos de informação são ligados com bibliotecas contendo:

- O conjunto de estatísticas de processamento relacionadas na célula.
- O conjunto de filas referenciadas pela célula.
- O conjunto de todos os planos e processos referenciados pelas especificações da produção.
- O conjunto de todas as ações de manutenção referenciadas pelos processos e transportadores.

Os dados são organizados em uma hierarquia. Dados ao nível do sistema são globalmente disponíveis, mas dados ao nível de fábrica são específicos para problemas particulares.

Cenários alternativos podem ser propostos para qualquer fábrica. A interface com o usuário é provida de um editor dirigido por menu. Existem dois menus funcionais com ajuda *on-line*. Através destes menus os conjuntos de dados podem ser montados. A exibição é criada de modo análogo pela seleção de elementos e posicionando-as sob o controle do cursor, assim, a utilização de um *mouse* é praticamente indispensável. A execução é supervisionada pelo segmento Simfactory do modelo. Uma representação animada da fábrica é fornecida e isto pode ser usado como *debugger*. Animação tem uma significativa sobreposição e pode ser removida para execuções de produção. Uma função de interrupção é fornecida com este propósito - o menu instantâneo (*snapshut*). Com este menu o usuário tem acesso a elementos dentro da simulação. Relatórios padrão incluindo informações da produção, das estações de processamento, do desempenho dos transportadores, das filas, da utilização dos recursos e consumo de materiais podem ser fornecidos em intervalos ou ao final da corrida.

O modelo genérico do Simfactory é escrito em Simscript II.5 e as características de aplicação-específica são selecionadas pelos dados de entrada. Esta abordagem permite ao fornecedor do software adequar seu pacote às necessidades particulares de um grupo limitado, simplificando a *interface* entre o modelador e o software. A característica do pacote que permite que um elemento seja dividido em outros, como uma folha de metal formando a base de várias moldagens, é valiosa e pertinente para a maioria das tarefas de modelamento da produção.

Com o desenvolvimento da linguagem original, Simfactory

também evoluiu. Por exemplo, está previsto uma versão fornecendo animação por meio de terminais gráficos ligados a mainframes. A plataforma computacional mínima é um PC-AT.

8.2.3. SAME

Uma companhia automobilística, a automação Renault, é responsável por este pacote que apresenta muitas semelhanças com o Simfactory, no que diz respeito a interface com o usuário. Ambos descrevem o modelo como uma série de blocos cujos parâmetros são especificados via formas que são apresentadas como menus *pull-down*. Estes blocos removem peças de um conjunto de filas e as empuram, algumas renomeadas, para outras filas. Internamente estes blocos possuem conceitos de eficiência, manutenção, tempo de *setup* e todas as demais características particulares de um ambiente de produção.

Same está disponível em vários módulos: Same/AGVs, para sistemas de transporte automatizados, Same/FAS, para armazenagem automatizada e Same/FMS, para sistemas flexíveis de manufatura e possui uma versão para computadores pessoais. O que os dois pacotes, Same e Simfactory, ainda possuem em comum é o refinamento para o exercício de construção do modelo representado pela animação.

8.2.4. Witness

Witness é um pacote projetado na Inglaterra com o nome de Forssight e comercializado no Estados Unidos com o nome Witness. É um simulador dirigido por dados criado para o estudo de sistemas de manufatura. Witness pode ser integrado com sub-modelos See-Why, se suas capacidades internas são insuficientes para uma tarefa em particular. Os elementos de modelamento do Witness são os relacionados com a fábrica - peças, máquinas, *buffers*, esteiras e mão-de-obra. Neste aspecto, o sistema apresenta pequenas diferenças de outros que têm sido discutidos, ou que seguem. O projeto de interface pode evoluir em paralelo e independentemente de modelos assemelhados.

Witness é um sistema de simulação da manufatura que incorpora características de Inteligência Artificial como forma de permitir que não especialistas construam modelos de sistemas

complexos de manufatura [26]. As referências [38], [68], [85] e [181], ilustram aplicações do pacote a sistemas de manufatura.

8.2.5. Xcell

O pacote Xcell procura tornar a simulação acessível a não profissionais incorporando a estrutura de trabalho das planilhas eletrônicas. O modelador tem uma única grade retangular de células uniformizadas, em que pode colocar um componente selecionado de um conjunto de 5 tipos: centros de trabalho, *buffer*, área de recepção, área de despacho e instalações de manutenção. O software tem um conjunto de funções chaves de controle e o processo de construção do modelo é controlado pela seleção da função apropriada dentre as seguintes opções:

- Fábrica nova (limpa o espaço de trabalho corrente),
- Projeto (uma nova fábrica),
- Análise (faz uma checagem da consistência do modelo),
- Execução (executa o modelo corrente),
- Gerenciador de arquivos (armazena e recupera o modelo),
- Muda a exibição (altera o conteúdo do monitor).

A seqüência de construção do modelo para se criar uma nova fábrica, inicia com a limpeza da tela. O cursor pode ser movimentado pela planilha e ícones selecionados são, então, colocados nas células. Os parâmetros dos ícones devem ser estabelecidos como, por exemplo, os tempos de serviço para centros de trabalho ou o nível de estoque de uma peça particular dentro de um *buffer*. Os caminhos através dos quais os trabalhos são executados são, então, definidos. As opções de análise checam as inconsistências do modelo, causadas por omissão, e estimam a capacidade do sistema na ausência de variabilidade. '*Runs*' executa o modelo. A execução pode ser interrompida para atribuir, esboçar, desenhar ou representar graficamente opções associadas com ícones. O tempo de execução pode ser escalonado, a velocidade variada e a duração recomposta em qualquer tempo.

Da mesma forma que o Simfactory, este pacote é realmente um modelo genérico dirigido por dados. Quando o usuário coloca componentes na planilha ele está, de fato, identificando entidades existentes. Cada componente já está inicializada por atributos *default*. O usuário, subseqüentemente e seletivamente, poderá alterá-los, mas isto só pode ser feito através de uma tabela de

atributos sob o controle de um editor, de modo que o sistema possa assegurar a validade de suas ações. Isto não somente criará um modelo que sempre rodará, mas também permitirá ao usuário mudar os valores dos parâmetros durante uma pausa na execução.

8.2.6. Modelmaster

O desenvolvimento do Modelmaster reflete em alguns aspectos a confusão que muitas vezes está associada à computação dentro da indústria. Constatou-se que dentro de muitas plantas de manufatura da General Eletric (GE) a simulação estava se tornando mais e mais popular [90]. Este fato foi notado pela variedade de softwares de simulação que foram encontrados em uso. Feita a constatação, uma equipe relativamente pequena de pessoas habilitadas começou a resolver o problema pelo desenvolvimento de um sistema único de simulação. A equipe estabeleceu que o sistema devia:

1. Ser abrangente, porque os ambientes de manufatura da GE são bem diversificados.
2. Ser simples de usar.
3. Livrar o usuário da programação.
4. Ter o máximo de gráficos coloridos para I/O.

Desta forma, o pacote de simulação Modelmaster foi projetado para permitir que pessoas, com pouco ou nenhuma experiência em simulação computacional, criassem e simulassem modelos de sistemas complexos de manufatura assegurando, ainda, que isto fosse feito rapidamente e eficientemente com um risco mínimo de que um modelo incorreto fosse criado.

O pacote, que foi originalmente planejado para ser usado somente dentro da GE, constitui-se de 4 módulos de simulação altamente flexíveis. Sendo um para processamento serial, um para ciclos de montagem, um para *job shop* e um para sistemas de movimentação de materiais. Estes módulos são únicos, no sentido de que todos incluem um ambiente não-programável, usuário-amigável, suportando entradas gráficas e orientadas por menu.

Destes quatro módulos básicos, sistemas híbridos foram formados evoluindo para aplicações mais gerais. Por exemplo, combinando os módulos de processamento serial e ciclos de montagem obtém-se um sistema particular apropriado para simular linhas de de montagem. Semelhantemente, combinando os módulos *job shop* e de

movimentação de materiais, resulta um sistema apropriado para simular SFM.

Após ter sido utilizado dentro da GE por 18 meses, entendeu-se que uma versão comercial poderia ser desenvolvida sem muito esforço. Trabalho que foi executado, resultando no lançamento do software como um pacote de simulação para microcomputador, ao final de 1985. Uma nova versão do pacote foi posteriormente lançada, contendo as seguintes funções:

1. *Layout* gráfico orientado por menu.
   - Entrada e saída do sistema;
   - Posicionamento de estações de trabalho;
   - Posição do *buffer* de armazenagem;
   - Tipos de peças e suas seqüências;
   - Múltiplas operações por peças;
   - Caminhos e passagens de transportadores;
2. Tipos e formas de entrada de dados.
   - Lotes e divisão de peças;
   - Montagem de tipos de peças;
   - Lotes e refugos probabilísticos;
   - Gerenciamento de recursos e turnos;
   - Tempos de máquinas ociosos planejados e também não planejados;
   - Chegadas de matérias primas programadas;
   - Sistemas de transporte múltiplos;
   - Cálculos automáticos de tempos de transporte;
   - Disciplina de filas automática (*fifo*, *lifo*, etc.);
   - Restrições de capacidade das filas;
   - Operações de *setup*;
3. Programa e relatórios da simulação.
   - Especificação da duração da corrida;
   - Lotes para análise estatística;
   - Remoção automática de dados espúrios iniciais;
   - Observação de períodos específicos de tempo;
   - Capacidade média, mínima e máxima de filas;
   - Utilização média, mímina e máxima das estações de trabalho;
   - Utilização de recursos de transporte;
   - Lotes de execução múltiplas;
4. Saídas gráficas

. Gráfico de barras da capacidade de filas;

. Gráfico de setor da utilização de recursos;

. Gráfico de Gant da utilização de estações de trabalho.

. Animação do sistema.

Pela lista acima, pode-se perceber que Modelmaster é um pacote bastante abrangente. A GE assegura que o modelo foi de grande valor no projeto de mais do que doze de suas grandes plantas [90].

8.2.7. See-Why

See-Why foi desenvolvido, na Inglaterra, em 1981, no departamento de Pesquisa Operacional da empresa automobilística *Britsh Leyland*. O pacote é baseado nos conceitos de simulação interativa visual, desenvolvidos por **Hurrion** [104], da Universidade de Warwick.

O software pode auxiliar no projeto de sistemas flexíveis de manufatura. As aplicações incluem analisar o uso de trabalho humano ou de robôs, implementação de troca de ferramentas, determinar o tamanho dos pontos de armazenagem, a velocidade de esteiras, etc.

A qualquer tempo, o usuário pode parar a execução para manipular o modelo e ver os efeitos imediatamente. O modelo uma vez ligado (*link*) pode ser executado sem reconstrução ou recompilação. Até 60 corridas de simulação podem ser armazenadas em disco a qualquer ponto da execução.

Embora o pacote tenha sido projetado para uso, principalmente, por especialistas, pessoas não especializadas podem fazer experimentos com modelos complexos com pouco treino.

O sistema opera com 3 níveis de interação *prompt-drive*:

. Interação geral para especialistas e não especialistas executarem tarefas, tais como, controlar a execução da simulação e reformatar a exibição da tela;

. Interação utilitária para especialistas examinarem e alterarem dados do modelo para correção, validação, experimentação e entradas e

. Interação do usuário, escrito por especialistas para especialistas e não especialistas em situações onde não são aplicáveis às interações do sistema [218].

O pacote possui um pré-processador de códigos denominado

Express (ver secção 9.1.5), desenvolvido para facilitar a construção do modelo.

See-Why está sendo substituído pelo Witness na modelagem dos vários tipos de sistemas, mas aqueles que precisam de um pacote de programação livre e aqueles que desejam trabalhar com Fortran permanecem fiéis [39].

8.2.8. Genetik

Um outro pacote que procura livrar o usuário da tarefa de programação é o Genetik. Como See-Why e seu predecessor Optik, Genetik é um pacote visual interativo. A principal característica da modelagem com este pacote é sua construção por unidades, tais como: unidade de dados, *pictures*, lógica, estatística, etc. Cada unidade é criada através de menus na tela e na medida que é criada é checada e convertida num código executável. Isto significa que não apenas o modelo é compilado sem demora, mas que muitos tipos de interação com o modelo são possíveis durante uma execução, isto é, alterações que de outro modo precisariam ser escritas em Fortran e o modelo recompilado. Neste sentido, Genetik funciona como um modelo genérico, mas tem a vantagem de que o usuário não fica restrito a lógica (regras de decisão) construídas do modelo genérico pelo seu programador [39].

8.2.9. Automod

É um produto da *Auto Simulations Inc*, cujo objetivo é a análise e projeto de fábricas. Possui estruturas para definir sistemas de movimentação de materiais com animação em 3-D, que é executada através de um pacote auxiliar denominado **Autogram.**

Interfase é um software de simulação da mesma família e pode ser classificado como um simulador orientado para o planejamento e a programação da produção, possuindo animação e saídas gráficas. Interfase integrado com Automod possilita ao usuário testar a resposta de sistemas complexos de movimentos de materiais e ver os efeitos combinados.

Os dois produtos constituem uma ponte para a lacuna (*gap*) entre a modelagem analítica e a operacional e incluem:

- Integração com base de dados corporativas.
- Integração com sistemas CAD.

. Programação.

. Animação 3-D.

. Simulação cinemática (movimentos mecânicos) de robôs e máquinas ferramentas.

. Gráficos comerciais.

Automod é escrito em GPSS e inclui biblioteca de lógica GPSS para equipamentos de manufatura e estratégias de controle. O usuário pode modificar e estender a lógica GPSS como o necessário. Se animação for desejada, um arquivo **localiza** é criado e registra todos os eventos do sistema como eles ocorrem através do tempo. Os componentes de manufatura em Automod possuem modelos geométricos 3-D, correspondentemente, definidos em Autogram. A animação 3-D do arquivo **localiza** é gerada num equipamento gráfico do tipo CAD [39].

8.2.10. Hocus

HOCUS (*Hand Or Computer Universal Simulator*) é um produto de propriedade do grupo de consultoria P-E da Inglaterra. Para utilizar este software o programador deve formular o modelo em termos de um diagrama do ciclo da entidade (DCE).

Um programa fonte em HOCUS consiste de uma descrição do diagrama bloco por bloco. O próprio HOCUS é um grande programa fornecido como código objeto. O modelo HOCUS é, desta forma, um arquivo de dados que é submetido ao programa.

Um programa HOCUS, inclui as seguintes secções:

1. Entidades. Uma lista das classes de entidades na simulação mostrando o número de elementos em cada classe e o número de atributos. Assim:

Emáquina 10 3

Erobô 2 4, indica que existem 10 máquinas com 3 atributos cada e 2 robôs com 4 atributos cada.

2. Filas. Uma lista de nomes de filas, denominados estados mortos, indicando todos os tamanhos máximos.

3. Campo de dados. Para a entrada de distribuições, geração de amostras, etc.

4. Atividades. Estabelece as condições necessárias para iniciar cada atividade, denominadas de estados ativos e as mudanças de estado resultantes. Ambos devem ser expressos em termos de filas (estados mortos) e entradas/saídas para cada estado ativo.

5. Histogramas. Especifica aqueles que serão utilizados para coletar os dados de saída.
6. Condições iniciais. Estabelece como se iniciará a simulação.

As opções de manipulação de atributos são limitadas. Processos complexos podem somente ser adequadamente modelados com a utilização de funções ou sub-rotinas e isto requer a escrita de código, que HOCUS é projetada para evitar.

O controle de tempo (relógio) inteiro do Hocus pode apresentar problemas no modelamento de sistemas que requerem modelamento preciso (emulação) e quando operações têm durações significativamente diferentes. Por exemplo, a troca de *pallets* ou movimentos de AGVs pode levar poucos segundos, enquanto operações de usinagem levam vários minutos [69].

As vantagens deste software são a facilidade de aprendizagem, uma vez que ele praticamente não requer conhecimentos de computação e a entrada orientada por menu, que o torna fácil de utilizar [163]. As desvantagens são a dificuldade em modelar sistemas que não se adaptam bem ao DCE e mesmo que o sistema possa ser modelado com sucesso por DCE, HOCUS não fornece maneiras do usuário editar o código Fortran, dificultando o tratamento de algumas complexidades de modelagem.

Constantemente atualizado com o avanço da computação o software pode rodar em computadores que suportam o sistema operacional Xenix, tais como os PC/AT. Se um processador gráfico estiver disponível, então Hocus pode ser utilizado em modo gráfico interativo. Utilizado de forma interativa, o pacote fornece uma maneira rápida de desenvolver simulação de sistemas que são modelados por DCE.

8.2.11. Grasp

O GRASP (*Graphical Robot Applications Simulation Package*) foi desenvolvido no departamento de Engenharia de Produção da Universidade de Nottingham, Inglaterra. É um sistema computacional para modelar e simular estações industriais de robôs.

No Grasp robôs são classificados de acordo com sua estrutura de articulação (*linkage*). Modelos de robôs específicos são construídos por procedimentos facilitados de entrada de dados. O corpo do robô é modelado utilizando formas primitivas, tais

como, cubos, prismas regulares e irregulares, cilindros e poliedros. Da mesma forma é construído o local de trabalho do robô. Estruturas permitem o rearranjo do local de trabalho facilmente. Em resumo, Grasp permite:

- Projetar o ambiente de trabalho do robô.
- Projetar as tarefas do trabalho do robô.
- Usar animação para ilustrar as operações de trabalho propostas.

Os passos típicos que devem ser executados pelo simulador são:

- Construir o modelo do local de trabalho.
- Checar se os ítens (robôs, máquinas, esteiras, etc.) adaptam-se ao ambiente.
- Checar se o robô pode alcançar locais críticos.
- Definir um caminho.
- Checar colisões.
- Produzir um processo.
- Produzir um programa.

O sistema é, desta forma, um método de simulação sob controle do projetista. Grasp provou ser apropriado para modelar as seguintes aplicações:

- Soldagem.
- Manuseio.
- Simular avaliações de segurança.
- Simular um SFM [24].

8.2.12 Simnek

É um pacote de simulação objeto orientado, do laboratório de produção NTH-SINTEF de Trondhein, Noruega, e que objetiva simular o projeto, instalação, operação e modificação de sistemas de manufatura.

Apresenta cinco funções principais. A função:

Modelagem do layout que representa o sistema real a ser simulado em um modelo computacional;

Modelagem do fluxo da entidade que representa dados, restrições de produção, planos de processos, especificações de produtos e sistemas de carga;

Simulação que executa a simulação atual. Este é o principal módulo de processamento de dados do sistema;

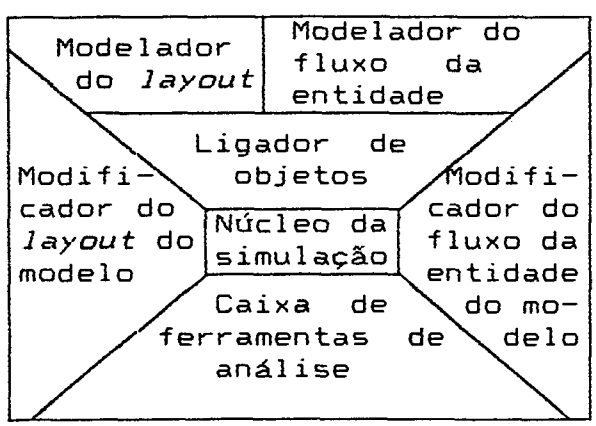
Análise de dados que é utilizada após a execução da simulação para apresentar dados em gráficos tais como de barras, setores, etc;
Implementação do modelo que age sobre os dados resultantes indicando possíveis conexões entre observações e parâmetros do modelo.

O sistema consiste de cinco módulos, todos implementados em C, que são: modelador de *layout*, modelador do fluxo da entidade, ligador (*link*) de objetos, núcleo da simulação e *kit* de análise. Um sexto módulo de suporte à decisão está previsto para integrar o sistema. A estrutura do simulador e dos seus módulos podem ser visualizados na figura 7.1.

O simulador Simmek apresenta uma abordagem objeto orientada da simulação, através dos seguintes grupos de objetos envolvidos na simulação de sistemas de manufatura: recursos, entidades e objetos necessários.

Os objetos recursos envolvem máquinas, armazenamento e unidades de transporte. Os objetos entidades representam as entidades fluindo no sistema. As peças que são movidas em todas as direções são tradicionalmente modeladas por meio de gráficos de fluxos padrões. Em muitos sistemas isto é uma representação conveniente mas podem, muitas vezes, ocultar fatos interessantes sobre o estado do sistema. Tais fatos podem ser revelados pela representação do fluxo da entidade por seqüências de objetos. Os objetos necessários podem ser considerados como um subgrupo dos objetos recursos caracterizados por serem consumíveis [169].

Figura 7.1 - Os módulos do simulador Simmek



Fonte: RAMSLI, 1989

8.2.13. Outros simuladores de manufatura

RTSC 250 (*Real Time Conveyor Simulator*) é um simulador de esteiras em tempo real e é o único software que tem interface direta com controladores lógicos programáveis para operar estes tipos de mecanismos de movimentação de materiais. Uma animação 2-D mostra em uma tela colorida os ítens fluindo através da esteira e o estatus dos sensores, interruptores, etc. O *layout* físico da esteira e outros parâmetros do sistema são definidos por tabelas no computador principal e passadas para o RTSC. Este simulador é acompanhado do hardware [88].

Fator é um produto da *Pritsker Corporation* baseado em simulação a eventos discretos com o objetivo de auxiliar o fornecimento de estratégias de manufatura e selecionar a melhor política para satisfazer os objetivos de produção a curto prazo. Programação da produção é a saída principal. Este software está disponível para simulação em tempo real utilizável nos modos de animação, simulação, emulação e monitoramento

ProModel é um simulador de sistemas de manufatura variando de pequenos *jobs shops* a grandes produções em massa e SFM. O pacote tem uma biblioteca de sub-rotinas que permite a modelagem simplificada de um grande número de características da manufatura. É um simulador parametrizado. Para casos especiais que não podem ser modelados através das sub-rotinas, ProModel fornece uma linguagem procedural completa com atributos, variáveis globais, variáveis de estado, variáveis lógicas. Para decisões muito complexas o usuário pode representar a lógica necessária em rotinas especiais escritas em C ou Pascal [39].

8.3. MODELOS GENÉRICOS DE SISTEMAS FLEXÍVEIS DE MANUFATURA

Vários esforços têm sido feitos para desenvolver capacidades gerais de modelagem e simulação para auxiliar no projeto e análise de SFM. O produto final destes esforços é, normalmente, um pacote usuário-amigável que permite a alguém com pouco conhecimento de simulação computacional especificar o sistema sob investigação e obter as medidas de desempenho pertinentes com relativa facilidade.

Os sistemas flexíveis de manufatura são complexos e

constituídos de muitos subsistemas. Encontrar um pacote que se adapte a situação de interesse sendo analisada nem sempre é fácil. Os capítulo 2 e 3 forneceram os elementos componentes, os subsistemas, bem como, a problemática que envolve o projeto, planejamento, programação e controle destes sistemas, com o objetivo de orientar a escolha do software mais adequado e também de fornecer subsídios para a construção de um software, sob-medida, próprio. Aqui é feita uma análise dos pricipais softwares específicos para simular os sistemas flexíveis de manufatura.

8.3.1. Gcms

GCMS (*General Computerized Manufacturing System*) foi o primeiro programa deste tipo desenvolvido para SFM. O pacote foi criado na Universidade de Pardue e originou vários outros pacotes, tais como MAST e GFMS. Sub-rotinas em Fortran inteiramente genéricas podem ser acessadas. Não existe distinção entre, por exemplo, uma estação de usinagem, moldagem ou mesmo inspeção. A única hipótese feita pelo GCMS é que todas as ações do sistema que são importantes para seu desempenho são descritas por uma seqüência ou subconjunto das oito rotinas básicas seguintes:

1. Operação em uma peça na estação;
2. A finalização da peça na estação;
3. Movimentos de peças para filas de peças completadas se existir;
4. Chamada para um sistema de movimentação de materiais, se existir no sistema, senão, coloca a peça em uma esteira;
5. Movimento de uma peça para um aparelho de movimentação de materiais que não uma esteira;
6. Deslocamento do sistema de movimentação de materiais levando a peça para a próxima estação;
7. Movimento da peça para a área de espera da próxima esteira, se existir;
8. Movimento da peça da área de espera na estação de processamento quando a estação está disponível e a peça satisfaz o critério de seleção.

Um outro tipo de dado de entrada para o pacote é a velocidade do sistema de movimentação de materiais. As 'equações' de movimento destes aparelhos são construídas e o programa precisa

saber a velocidade média e a distância entre as estações para calcular o tempo de viagem [23].

8.3.2. Mast

MAST (*MAnufacturing System design Tool*) é um simulador para projetar um SFM envolvendo máquinas ferramentas. Mast pode representar uma variada gama de componentes de hardware e algoritmos de controle de processos. O pacote utiliza um sistema de arquivamento computacional de formato livre. Bibliotecas de algoritmos estão disponíveis para métodos alternativos de controle do software.

Mast produz três tipos de relatórios do sistema sendo modelado, cada um apresentando os resultados listados abaixo:

1. Descrição do sistema e dos dados de entrada:
   - Descrição do tipo de peças,
   - Descrição da estação,
   - Descrição dos *pallets*,
   - Descrição dos tipos de veículos,
   - Descrição do tipo de *layout*.
2. Estatus do sistema:
   - Estatus inicial,
   - Estatus das peças,
   - Estatus das estações,
   - Estatus dos veículos,
   - Estatus dos *pallets*,
   - Estatus dos caminhos.
3. Relatório resumo do desempenho:
   - Peças completadas,
   - Desempenho dos tipos de peças,
   - Estatísticas dos *pallets*,
   - Operações executadas,
   - Desempenho das estações,
   - Desempenho dos *shuttle* das estações,
   - Desempenho dos veículos e
   - Desempenho dos caminhos.

O pacote foi originalmente escrito em GASP, mas este código base foi substituído. MAST é escrito em Fortran e pode rodar em qualquer mainframe ou minicomputador. Consiste de aproximadamente 7000 sentenças e pode ser usado com muitos

compiladores Fortran. MAST pode ser complementado com os módulos SPAR, e BEAM. SPAR (*System Planning of Aggregate Requeriments*) fornece um instrumento sob medida pela qual o desempenho estático do projeto de um sistema pode ser acessado para assegurar que um esboço do projeto faça sentido. SPAR também cria um arquivo de dados de entrada para o MAST. BEAM (*Background and Enhanced Animation for Mast*) fornece saídas gráficas na forma de diagramas e gráficos. **Lenz** [160] sugere o uso dos três pacotes (MAST, SPAR e BEAM), como um conjunto apropriado para o projeto e avaliação de SFM. SPAR permite ao usuário definir o sistema e também faz uma análise estática de capacidade assegurando, desta forma, que os níveis de recursos propostos, por exemplo o número de AGVs, seja razoável. MAST simula e BEAM fornece saídas animadas.

As capacidades do pacote cobrem o hardware e o sistema de controle. As capacidades de hardware incluem:

1. **Famílias múltiplas de peças.** A descrição da peça em MAST consiste de informações de produção e da rota da peça. Esta rota contém uma lista das operações e as estações onde eles serão executadas.
2. **Tipos de estações.** Cada área de carga e descarga, máquina ferramenta , estação de inspeção ou qualquer outra estrutura que executa uma operação, é representada como uma estação MAST.
3. **Transportadores.** MAST admite vários tipos de mecanismos de manuseio de materiais, incluindo esteiras, carros puxados por correntes, carros dirigidos por fios e guindaste suspenso.
4. **Estruturas de armazenagem em processo.** Cada sistema deve ser capaz de acomodar peças em-processo e armazená-las temporariamente até que a próxima estação torne-se disponível. Mast pode estudar estruturas alternativas de armazenagem como filas de *pallets* em cada estação, armazenagem no transportador ou estruturas dedicadas em processo que armazenam peças longe das estações para aliviar congestão ou bloqueios.
5. **Layout do sistema.** MAST permite tanto sistemas de ciclo fechado (*closed-loop*) quanto linhas de transferência contínuas (*straight transfer line*). O *layout* do sistema é representado separadamente do tipo de transporte e pode incluir múltiplos caminhos entre estações. MAST gera todos os caminhos possíveis entre a estação de origem e a de destino e o controle avalia cada caminho de acordo com algoritmos selecionados pelo usuário.

As capacidades de controle incluem:

1. **Balanço de peças.** MAST pode controlar aumentos repentinos para prevenir o desbalanceamento entre a capacidade da estação e a requerida. O balanço de peças pode ser simulado permitindo um número fixado de *pallets*. Uma regra de introdução controla quando e que tipo de peça são introduzidas no sistema.

2. **Seqüência de operações.** MAST pode investigar seqüenciamento de operações fixo ou dinâmico. Para seqüências fixas o software faz o roteiro das peças de acordo com a ordem da operação descrita como dado de entrada. No seqüenciamento dinâmico a ordem das operações é determinada por uma regra de seleção que usa dados, tais como, disponibilidade da máquina, congestionamento do transportador e distância a ser percorrida.

3. **Seleção da estação.** O software pode investigar critérios alternativos para selecionar estações. Se somente uma estação pode executar uma operação, nenhuma regra será necessária. Quando mais de uma estação pode executar uma operação, a próxima estação poderá ser selecionada pela distância de transporte, pelo *estatus* (vazia ou ocupada) ou prioridades definidas pelo usuário.

4. **Controle de transporte.** MAST pode investigar transportadores endereçáveis e não endereçáveis. No caso de endereçáveis, ele ou seleciona os componentes para transportar uma peça ou organiza e controla os movimentos de todos os componentes. MAST simula o movimento de cada transportador considerando a congestão do tráfego e não permite ultrapassagens por transportadores individuais. A regra de seleção para os não endereçáveis permite que a peça deixe a estação imediatamente sem congestionar o transportador.

5. **Controle de armazenagem em-processo.** MAST pode simular sistemas de manufatura com qualquer combinação de armazenagem em-processo de peças no transportador, nas estações ou em filas especificadas. A regra de decisão para armazenagem em-processo deve estar de acordo com o hardware.

6. **Confiabilidade.** MAST pode simular falhas e reparos de estações, transportadores e secções de esteiras. Falhas e reparos podem ser gerados aleatoriamente de acordo com distribuições de probabilidade selecionadas pelo usuário ou podem ser controlados por dados de entrada.

O pacote é propriedade da CMS Research Inc e é

comercializado na Inglaterra, pelas indústrias Citroen. MAST está sendo expandido para fornecer animação gráfica. A partir de 1983 tornou-se disponível uma versão para microcomputador [128].

8.3.3. Map/I

MAP/I (*Modelling and Analysis Program*) é um programa de modelamento e análise, desenvolvido pela empresa Pritsker e Associados e utilizado para projetar e avaliar sistemas de manufatura de lote em que peças individuais são processadas discretamente em estações de trabalho e são transportadas em quantidades iguais ou maiores que uma. O pacote pode ser usado para analisar *job shops*, *flows shops*, linhas de montagem e sistemas de manufatura computadorizados com robótica e SFM. Também pode ser utilizado para investigar configurações alternativas do sistema e procedimentos operacionais.

Um sistema de manufatura de lotes é um conjunto integrado de máquinas, pessoas, materiais e equipamentos de transporte que produzem peças. Existe um procedimento em MAP/I para representar cada um destes componentes maiores do sistema de manufatura de lote.

Estações são definidas como locais onde ocorre o processamento. Estações podem ser uma única máquina, um grupo de máquinas semelhantes, um local de trabalho onde uma operação manual é executada ou uma área de carga e descarga de fixadores. Estações trabalham em um dos três seguintes modos: processando peças individuais, produzindo múltiplas peças de uma única ou montando uma peça de várias. Estações, normalmente, possuem espaços disponíveis para peças esperando processamento (armazenagem pré-processo), bem como, para peças prontas esperando transporte (armazenagem pós-processo). Quando estas áreas estão cheias elas afetam a operação do sistema, bloqueando a atividade até que o acúmulo seja retirado.

Transportadores são os aparelhos que movem as peças entre as estações. Dois tipos de transportadores podem ser modelados por MAP/I. O primeiro tipo, denominado transportadores regulares, incluem empilhadeiras, guindastes, carros e robôs. O segundo tipo de aparelhos de transporte que podem ser modelados são as esteiras. Transportadores regulares são unidades individuais que operam independentemente, podendo servir várias

estações e transportar peças em lotes de tamanhos de um ou mais. Esteiras são freqüentemente utilizadas em sistemas de manufatura com maior grau de automação para transportar peças entre estações e podem servir como pontos de armazenagem temporária.

Em muitos sistemas de manufatura, pessoas são necessárias para operar máquinas e equipamentos de transporte ou para executar operações manuais, tais como, inspeção ou uma correção. Em alguns sistemas, operadores são recursos críticos e sua disponibilidade pode afetar grandemente o desempenho global do sistema. Tipicamente, as pessoas que trabalham em sistemas de manufatura são divididas em classes com tarefas específicas. Uma destas classes pode ser a dos motoristas de empilhadeiras e uma outra, pode ser o operador de máquinas. Quando um operador é necessário para uma tarefa, uma pessoa específica é selecionada de uma das classes.

Fixadores são mecanismos que prendem uma peça ou grupo de peças durante o processamento ou transporte. Um *pallet* pode ser pensado como um fixador. Normalmente, um fixador é usado por uma peça por várias operações consecutivas de processamento e subseqüente transporte. Após finalizado um conjunto de operações, a peça pode ser virada ou rotacionada no fixador antes de sofrer o próximo conjunto de operações. Uma peça pode requerer o uso de vários fixadores até completar seu processamento. Contrariamente, um tipo de fixador pode ser necessário por várias peças diferentes. Devido aos seus altos custos, fixadores muitas vezes são uma restrição em sistemas de manufatura. O número de cada tipo de fixador disponível, normalmente, determina o número máximo de peças que podem estar simultaneamente no sistema, bem como o *mix* corrente de tipos de peças. As regras utilizadas para alocar fixadores para os vários tipos de peças nas estações de carga têm um grande impacto na operação do sistema [139].

A simulação da manufatura de lotes (SML) transforma matéria-prima em peças finais processando o material nas várias estações no sistema. Cada tipo de peça tem um fluxo operacional pelo qual ela é feita. Peças são caracterizadas pela seqüência dos passos do processamento necessário para produzí-las. Uma única seqüência de processamento deve ser especificada para cada peça.

Uma vez descritos estes componentes básicos do SML, informações adicionais acerca de programação, operações de

substituição, paradas (quebras) de transportadores e estações são especificados. Isto é seguido pela simulação de informações específicas tais como: condições iniciais para o modelo, a simulação do tempo de partida e do tempo final e o número de corridas de simulação a serem feitas.

MAP/I fornece um conjunto pequeno de comandos facilmente aprendidos para representar os componentes estruturais principais do sistema sendo modelado. O usuário simplesmente combina os componentes estruturais na ordem correta para representar este ou aquele sistema e então parametriza cada componente. Desta forma, as suas aplicações são limitadas, mas o desenvolvimento de um modelo pode ser executado com uma rapidez não usual [14]. MAP/I está disponível, atualmente, somente para mainframes e é auxiliado pelo ambiente de suporte TESS [172].

8.3.4. Fmsds

FMSDS (*Flexible Manufacturing System Decision Support*) é um simulador de SFM, totalmente escrito em C, construído por **Wu** [230] como tese de doutorado na Universidade *Case Western Reserve*, EUA. É um software interativo, usuário amigável para auxiliar a tomada de decisão em projeto, planejamento e operação do SFM.

A principal característica do FMSDS, e a que o diferencia dos demais simuladores de manufatura, é a união de três ferramentas decisórias principais. Um modelo de redes de filas, um programa de simulação e uma heurística para *layout*. O modelo de redes de filas desenvolvido é uma extensão do MVA-Q e CAN-Q (ver capítulo 4). O software de simulação combina características encontradas em vários pacotes de simulação, incluindo animação. A heurística de *layout* desenvolvida é uma generalização do CRAFT.

Muitas regras de operação podem ser implementadas com o simulador FMSDS. Para uma família ou todas as peças, para uma ou todas as estações de trabalho, para um aparelho ou todo o sistema de movimentação de materiais. As três principais regras de operação são: regra de despacho (utilizada pela estação de trabalho para retirar peças da entrada da fila), regras de encaminhamento (utilizada pelas peças para decidir onde ir quando chegam a uma estação que não está funcionando), regra de seleção do aparelho de MM (utilizada por peças para decidir que transportador tomar, uma vez terminado o processamento corrente),

regra da seqüência de apanhamento (usada pelo SMM para decidir que peça apanhar primeiro), a estratégia da máquina quebrada (usadas pelas estações de trabalho para decidir o que fazer em caso de quebra de máquinas) e regra do balanceamento dinâmico de carga (usada por estações de trabalho para decidir como compartilhar dinamicamente suas cargas de trabalho).

O FMSDS oferece algumas heurísticas para usuários examinarem estratégias alternativas de balanceamento de carga, tais como: **hpf** (*highest priority first*), **aleatória**, **maxwl** (*maximum work load first*), **maxct** (*maximum cycle time first*), **maxq** (*maximum queue lenght first*), etc. e regras de ordem de apanhamento para sistemas de MM, tais como: **bsf** (*bottleneck station first*), **aleatória**, **hpf** (*highest priority first*).

Os resultados intermediários de qualquer ferramenta decisória (modelos de filas, heurísticas de *layout* e simulação), podem ser compartilhados com outras ferramentas dentro do FMSDS. Por exemplo, a construção da configuração para o modelo de simulação pode ser utilizada pelo modelo de rede de filas ou a heurística de *layout*. Por outro lado, os resultados da heurística para *layout* podem ser utilizados como uma entrada para a simulação. O usuário pode andar para trás e para diante entre as ferramentas instantaneamente, facilitando o processo de tomada de decisão.

Aplicações do FMSDS incluem [230]:

1. Projeção de custos para avaliar mudanças de configurações do sistema.
2. Pré-testes operacionais de configurações alternativas do sistema para decisões de:
   - Aquisição de novas plantas;
   - Substituição de equipamentos;
   - Adições de equipamentos;
   - Abandonar equipamentos;
   - Modernização de equipamentos;
3. Pré-testes operacionais de estratégias alternativas de controle para:
   - Mudanças de rotas de peças;
   - Ajuste das taxas do *mix* de peças;
   - Aumento ou decréscimo do número total de ítens;
   - Ajuste fino da velocidade do sistema de MM;

- Ajuste da capacidade dos *buffers*;

- Ajuste fino das regras operacionais (encaminhamento, reencaminhamento, seleção de aparelhos de MM, despacho, quebras e balanceamento dinâmico de carga).

8.3.5. Fmssim

FMSSIM (*FMS SIMulator*) é um simulador de propósito geral, evento-discreto, usuário-orientado, para manufatura flexível. Desenvolvido por **Elmaraghy** e **Ho**, em 1982, com o propósito de ser utilizado para testar projetos e para estudar os efeitos de estratégias de controle, regras de prioridades de programação e o *layout* de estações de trabalho. As estruturas (*building blocks*), que podem ser usadas para modelar o SFM são as estações, rotas e *shutles*.

Uma estação é uma estrutura do sistema onde uma operação é executada, uma rota é um caminho ao longo do qual as peças são deslocadas e um *shutle* é um *buffer* fila. A topologia do SFM pode ser definida por uma rede de pontos e rotas, representando locais onde mudanças de direção, ramificações ou transferência de peças entre trilhos e o *shutle* ocorrem [74].

Uma vez que o sistema e sua topologia estejam definidos, as peças, seqüências de operações, a confiabilidade das máquinas e as várias regras de decisão devem ser fornecidas pelo usuário para o modelo do sistema. O FMSSIM executa a simulação pela construção de um modelo apropriado de uma biblioteca de sub-rotinas que foram especificamente escritas para uso com este pacote.

As saídas da simulação são na forma de relatórios de desempenho e de animação gráfica. Relatório de desempenho inclui um resumo estatístico das medidas de desempenho do sistema, tais como: número de peças completadas, fluxo de tempo médio e máximo e a utilização de estações. A animação gráfica na forma de saída exibe o movimento das peças através do sistema. Esta espécie de saída pode ser muito útil na detecção de gargalos e para executar análise de sensibilidade.

8.3.6. Modular FMS simulator

É um simulador de propósito geral, usuário orientado que utiliza a abordagem do evento. Foi projetado para auxiliar o

projeto, planejamento e controle de sistemas flexíveis de manufatura. Foi também, a exemplo do FMSDS, originário de uma tese de doutorado, neste caso de **M. Montazeri** [143], em 1987, na Bélgica.

Este simulador foi desenvolvido com as seguintes características [144]:

1. **Modularidade.** O pacote consiste de um conjunto de diferentes módulos para cada tipo de atividade no sistema, que podem ser controlados independentemente. Para operar como um sistema completo basta ligar (*linker*) os módulos. Esta característica permite ao usuário escrever facilmente seus próprios módulos (se necessário) e adicioná-los ao pacote e também ignorar módulos que não o interessam.
2. **Amigabilidade.** O pacote é orientado por menu e a interface com o usuário é feita através de perguntas, em ordem lógica, como forma de especificar o sistema sendo modelado. Existe uma rotina diagnóstico para detectar possíveis erros de entrada.
3. **Regras decisórias.** Uma grande variedade de regras decisórias são fornecidades no pacote e podem ser utilizadas em cada ponto de decisão. O usuário dispõe da opção de escrever suas próprias regras de decisão e adicioná-las ao pacote, se desejar.

O modular FMS simulator foi projetado de forma a permitir a simulação de muitos tipos de configurações de sistemas, *layouts*, peças, componentes, regras operacionais, etc.

As principais entidades no modelo são peças, pallets, estações (máquinas), buffers e veículos. *Shutles* são considerados atributos das máquinas. A operação do SFM é vista como uma sucessão de eventos centrados nas peças a serem processadas. Foi escrito em Fortran 77 e GPSS e, inicialmente, implementado em minicomputadores com versões previstas para mainframes e PCs.

8.3.7. Outros simuladores de sistemas flexíveis de manufatura

GFMS (*General Flexible Manufacturing Simulator*) utiliza a linguagem Fortran-77 e GASP. Pode simular SFM com múltiplas estações de trabalho e de carga/descarga, conectadas por esteiras, caminhos ou robôs. Estações podem ter filas do tipo *shutle*. Falhas de estações, bem como congestão de sistemas de manuseio de materiais, também, podem ser modelados. Ferramentas não podem ser modeladas. O pacote GFMS foi desenvolvido pelo *Draper Laboratory*.

Inc. dos Estados Unidos [23].

SPEED utiliza a linguagem Fortran-66 e modela SFM em que peças são deslocadas em veículos, embora algumas esteiras também possam ser modeladas. Pode modelar, ainda, congestão de estações, máquinas, quebras, programação de operações e rotas probabilísticas para peças. Não requer programação e é inteiramente orientado por menu. Um gerador interativo pode ser utilizado para construir o modelo bem como para treinar novos usuários [23].

8.4. CONSIDERAÇÕES

Um grande número de softwares está disponível para simular SFM. Selecionar um, dentre eles, deve ser feito levando em conta a capacidade de modelar fielmente o sistema, a conveniência de uso, o tempo e o custo necessário para desenvolver o modelo ao invés de uma análise detalhada das características técnicas [39].

Estes pacotes constituem recursos mais sofisticados do que os softwares anteriores, exigindo pouca ou nenhuma programação. Muitos oferecem características adicionais como animação, interação com o usuário e entrada de dados facilitada. Em compensação, normalmente, custam caro, entre 5 e 100000 dólares, e oferecem uma flexibilidade menor do que as opções anteriores.

O preço que se paga pela facilidade de uso destes simuladores é uma considerável perda de generalidade na modelagem, visto que eles incorporam muitas hipóteses restritivas, este tipo de software adapta-se bem somente aos sistemas que satisfaçam estas hipóteses [211].

CAPÍTULO IX

9. PRÉ-PROCESSADORES, PROGRAMAS GERADORES E AMBIENTES DE SUPORTE

9.1. PRÉ-PROCESSADORES E PROGRAMAS GERADORES

Um programa gerador (PG) é um software que produz um programa em alguma linguagem objeto. É feito sob medida para ajustar-se a um conjunto específico de necessidades. Um PG, com a finalidade de tornar-se atrativo para o usuário, deve permitir uma considerável liberdade de expressão. Contudo, é essencial que o PG exerça algum controle sobre a especificação do problema, a fim de que inconsistências e omissões possam ser detectadas. Entradas para um PG devem, por essa razão, tomar lugar via um diálogo com o usuário no qual, dentro de uma estrutura imposta, ele possa responder em um formato razoavelmente livre e as saídas são, normalmente, em linguagem de alto nível.

Em resumo, um PG pode ser caracterizado como uma rotina que aceita uma única entrada descritiva de um modelo de simulação e produz um programa computacional equivalente em uma linguagem de simulação ou em uma linguagem de alto nível [136].

A partir destas características um programa gerador pode ser indistinguível de um pré-processador. Muitos softwares de simulação, caracterizados como linguagens ou pacotes são, de fato, pré-processadores Fortran ou de outras linguagens. Estritamente falando, o termo **programa gerador** ou, ainda, **gerador de programas** deve ser aplicado somente se as saídas podem ser produzidas em mais do que uma linguagem (ou dialeto), enquanto que o termo **pré-processador** deve ser aplicado quando as saídas são produzidas em uma única linguagem [164].

Embora possa ser creditado que o primeiro programa gerador foi a interface pergunta/resposta para a linguagem Simscript, ferramentas atuais originaram-se de trabalhos do início dos anos 70, como Caps e Draft [136]. Subseqüentemente, uma grande variedade de programas foram produzidos [131, 208].

A função de um programa gerador ou pré-processador é evitar algumas das dificuldades que são experimentadas por modeladores que procuram utilizar uma linguagem de programação

geral ou uma linguagem de simulação. Em essência, estas dificuldades surgem:

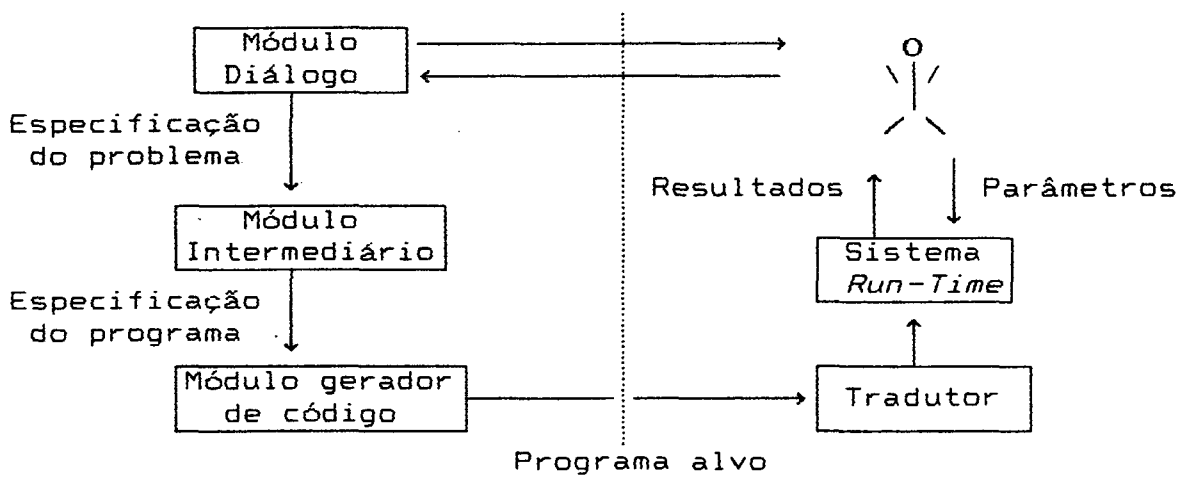
(1) Do tempo e esforço necessário para apreender uma linguagem geral ou de simulação, sua estrutura e sintaxe;

(2) De problemas práticos de executar o programa fonte, num computador disponível, que pode ser diferente do computador para o qual o programa fonte foi originalmente desenvolvido.

Um indíviduo sem ajuda especializada necessitará pelo menos três meses para obter proficiência na utilização de modelos Simula ou Simscript. Além disso, sabe-se por experiência que quando modeladores utilizam seus softwares com pouca freqüência, este software não é mantido com interesse [136]. Em tais circustâncias o modelador será obrigado a gastar tempo em problemas de programação de sistemas que estão essencialmente fora do âmbito de seu controle e interesse direto [136].

Um PG consiste de três componentes principais que são completamente distintas. O módulo diálogo, um módulo intermediário e o módulo de geração de código. As relações entre estes 3 módulos e o usuário estão na figura 9.1.

Figura 9.1 - Estrutura do programa gerador



Fonte: LUKER & BURNS, 1986

O módulo diálogo obtém do usuário as especificações da estrutura do problema a ser resolvido. Esta definição estrutural é, então, usada pelo gerador de código para produzir o correspondente programa alvo na linguagem desejada. Esta transformação fará uso de um módulo intermediário.

O programa alvo é, então, submetido a um tradutor apropriado junto com quaisquer dados necessários pelo programa para obter os resultados desejados [131].

9.1.1. Programas geradores e pré-processadores interativos

Por conveniência, muitos geradores operam em sistemas computacionais interativos e são conhecidos como **programas geradores interativos** (PGI). Os PGI são baseados no conceito de DCE. Para utilizar um PGI o usuário primeiro esboça um DCE tão completo quanto possível do sistema. Frente a um monitor o usuário é interrogado pelo programa gerador interativo em uma linguagem Inglês-semelhante. Com as respostas do usuário o PGI combina os ciclos de atividades das várias classes de entidades e produz um programa fonte de simulação em uma linguagem apropriada. Provavelmente o mais conhecido programa gerador interativo seja o **Caps**, que produz código **Ecsl** e **Draft** que está disponível em versões produzindo códigos em **Algol**, **Fortran** e **Simula**, entre outros. Na realidade, somente **Draft** é um PGI, **Caps** é tecnicamente um pré-processador interativo [163].

9.1.2. Draft

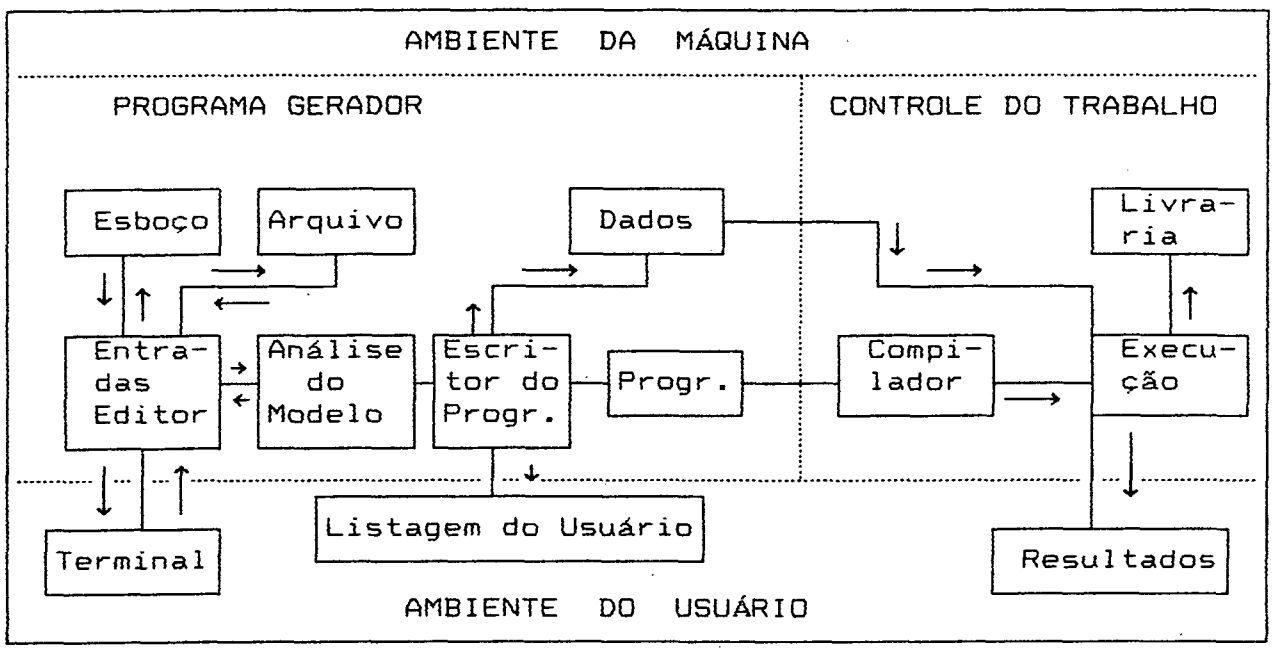
O modelo é expresso como um diagrama ciclo da entidade (DCE) introduzido via terminal. O módulo Entrada/Editor identifica erros menores ou semânticos (Como por exemplo: uso de nomes reservados de variáveis, nomes duplicados) e leva o usuário a entrar corretamente. Ele também cria uma cópia *'backup'* da entrada num arquivo temporário, que pode ser acessado mais tarde para modificações ou armazenado como uma cópia compacta do modelo.

O módulo de análise checa os erros de entrada, que podem ser muitas vezes corrigidos *on-line*, e prepara um arquivo internamente codificado das interações das entidades dentro do modelo. O arquivo forma uma entrada geral para o escritor do programa selecionado pelo usuário. É depois deste estágio que a descrição do modelo é transformada em uma abordagem particular de uma linguagem alvo que pode ser evento, atividade ou processo. O programa principal é, então, produzido. O programa gerador é capaz de controlar operações do computador principal, fornecer código livre de erros, que pode ser usado com um modelo para

melhoramentos adicionais, e oferece as vantagens e conveniências de uma notação taquigráfica (*shorthand*).

O programa gerador Draft é estruturado como um grupo de módulos funcionais visualizados na figura 9.2 Esta estrutura apesar de específica para o Draft é geralmente representativa de outros programas geradores. Draft fornece um código em Fortran e é um pré-processador para o pacote Simon [134].

Figura 9.2 - Estrutura do programa gerador Draft



Fonte: **STANDRIDGE, 1986.**

9.1.3. Caps

CAPS é um pré-processador para a linguagem **Ecsl** [163]. Da mesma forma que Draft, a entrada é uma tradução direta do diagrama ciclo de atividades. A aderência às regras para traçar a formulação do diagrama em um programa computacional assegura que o programa sempre será executado. Em contraste com esta abordagem **Birtwistle** e **Luker** [164], desenvolveram um interpretador intermediário para a abordagem do diálogo. Diálogos aceitam descrição do modelo na forma de respostas ao *prompt* gerado pelo computador.

Em Caps, o diálogo é dividido em seis secções [39]:

1. **Lógica**. Em que as classes de entidades são estabelecidas e a

lógica do sistema é descrita para o programa gerador Caps.

2. **Prioridades.** Estabelecendo se qualquer fila tem uma disciplina não *fifo* e escolhendo as prioridades das atividades.

3. **Aritmética.** Estabelecer as distribuições de probabilidades de onde as amostras serão extraídas e definir todos os atributos.

4. **Registros.** Decidir que dados de execução deverão ser coletados para as várias filas a fim de registrar o comportamento do modelo.

5. **Condições iniciais.** Estabelecer as condições de execução e outras condições iniciais.

6. **Administração interna.** Checar se qualquer palavra reservada do Ecsl foi utilizada sem perceber e sugerir melhoramentos para o programa.

9.1.4. Autosim

O grupo de Pesquisa Operacional da indústria do aço de Newcastle, Austrália, utiliza extensivamente a linguagem Simula, desde o início dos anos 70, com as principais aplicações sendo dirigidas para a simulação de plantas, portos e sistemas de transporte.

Apesar da vantagem em utilizar uma linguagem de simulação, o grupo encontrou dificuldades com o grande gasto de tempo em adquirir plena competência no uso da linguagem, tempo muito longo para o desenvolvimento de modelos complexos e dificuldade para manter os modelos já construídos. Surgiu, desta forma, a necessidade de uma abordagem mais estruturada para a codificação dos modelos de simulação, de forma a requerer pouca programação. Foi decidido, assim, desenvolver um gerador automático de simulação, Autosim (*AUTOmatic SIMulation*), baseado na linguagem Simula.

O primeiro estágio da abordagem do Autosim para desenvolver um projeto de simulação é construir um DCE. O segundo é codificar o DCE usando um conjunto de regras, razoalvemente simples e que permite a descrição, não só da estrutura lógica do diagrama, mas também de todos os cálculos, ramificações e instruções para produzir o programa final. O terceiro estágio é processar o arquivo contendo o DCE codificado por um programa Autosim (escrito em Simula) que gera um programa fonte em Simula. Este arquivo programa é, então, processado por um compilador Simula e executado utilizando arquivos de dados de entrada

apropriados.

Quando, inevitavelmente, mudanças são necessárias o DCE e o arquivo código são alterados e um novo programa Simula é gerado. No entanto, o código Simula é bem estruturado e legível e pode, como último recurso, ser modificado diretamente [225].

9.1.5. Express

Express é um pré-processador para o pacote de simulação interativa visual See-Why, escrito com a finalidade de livrar o usuário do pacote da necessidade de escrever em Fortran. O código é escrito numa versão simplificada da linguagem. Express checa cada linha para assegurar que foi corretamente convertida em Fortran. Existe uma rotina See-Why em Fortran para cada função em Express. O modelo é compilado em código executável como seria qualquer outro programa Fortran.

Express é um exemplo de um gerador de código para um pacote de simulação gráfico. Ele gera modelos See-Why, possuindo duas fases principais, o estágio *define* e o *events*. No estágio *define*, é definido cada elemento (entidade, conjunto, série temporal, etc.) e como ele será apresentado na tela. Características estáticas de fundo são, também, definidas. A fase *events* é mais rudimentar. Enquanto o programa gerador Caps obtém detalhes dos ciclos de atividade e gera um programa na linguagem alvo, Express não livra o usuário da necessidade de escrever o código lógico do modelo. Em vez disso, ele auxilia o processo permitindo escrever a lógica em 'pseudo' Fortran, checa a sintaxe de cada sentença interativamente, indica elementos não definidos, etc. e, então, converte a entrada em um programa Fortran de trabalho. A lógica em si não é checada [39].

9.2. AMBIENTE SUPORTE

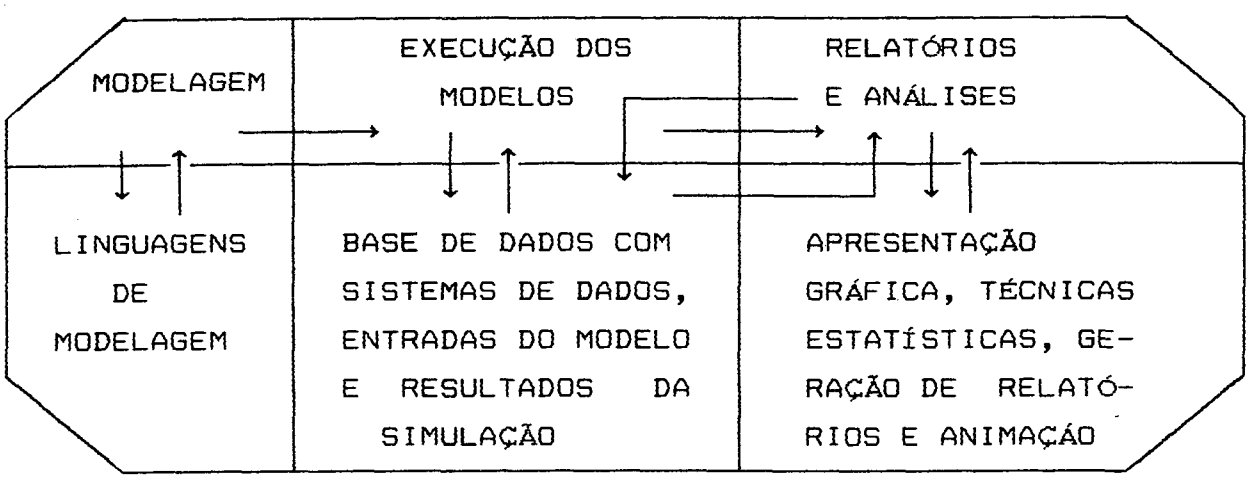
Um sistema completo de suporte é aquele que auxilia em todos os aspectos de estudo de simulação, através do fornecimento de módulos para a construção do modelo, controle de experimentos, análise e apresentação de resultados [163].

9.2.1. Tess

No início dos anos 80 foi reconhecido que o software deveria suportar outros aspectos do projeto de simulação além da codificação do programa. TESS (*The Extended Simulation System*) representou uma nova geração de softwares que integrou a construção do modelo, execução da simulação e análise e apresentação (inclusive gráfica) dos resultados.

A figura 9.3 mostra a estrutura para a linguagem e ilustra as áreas cobertas.

Figura 9.3 - Problemas resolvidos com TESS



Fonte: **STANDRIDGE, 1984.**

Acima da linha central estão os passos através dos quais o modelador prossegue. Abaixo da linha estão as estruturas de suporte fornecidas através de ferramentas de software. As flechas representam os caminhos de acesso às ferramentas. O objetivo é fornecer um sistema integrado.

Este pacote está disponível como um ambiente de suporte para Slam II, Map/I e Gpss/H. O controle global é feito por três funções: *Build*, *Report* e *Graph*, que operam nos vários elementos de dados dentro dos passos da tarefa de modelagem [209].

Tess fornece uma estrutura para executar estudos de simulação. Esta estrutura é concebida como uma matriz cujas colunas representam funções a serem executadas em um estudo de simulação e cujas linhas representam elementos do estudo sobre os quais as funções agem. Os elementos são agrupados em três categorias, conforme figura 9.4. Elementos de modelagem descrevem o sistema. *Facilities* são modelos esquemáticos usados para mostrar

a estrutura e as relações entre entidades. Redes são modelos de redes Slam II do sistema e definições são as características dos dados definidos pelo analista. Os elementos de simulação e análise como controle, dados, resumo e cenário fornecem especificações para executar a simulação, gerenciar a entrada de dados da simulação e os seus resultados e coletar estatísticas dos valores da simulação. Os elementos de apresentação formato e regras especificam como os valores de saída devem ser apresentados.

Figura 9.4 - Funções TESS

Elementos de		Funções	*Build*	*Report*	*Graphs*
	Modelamento	*Facilities*	*		*
		Rede	*		*
		Definição	*	*	
	Simulação e Análise	Controle	*	*	
		Dados	*	*	*
		Sumário	*	*	*
		Cenário	*		
	Apresentação	Formato	*	*	*
		Regras	*	*	

Fonte: **STANDRIDGE, 1985**

Os nove elementos são ligados por 3 funções. **Construção** (*Build*), **relatório** (*report*) e **gráficos** (*graphs*). Construção cria ou modifica a ocorrência de um elemento. Relatório é usado para exibir a ocorrência de elementos em um mecanismo alfanumérico tal como uma impressora. Gráfico exibe ocorrências de elementos em um mecanismo gráfico tal qual um terminal ou *plotter*.

Vinte combinações das funções e elementos são viáveis. Estas 20 combinações formam as bases da linguagem Tess, que é uma linguagem de comandos não procedural para especificar operações. Algumas combinações não são permitidas. As possíveis estão assinaladas com um asterisco.

Uma característica importante deste sistema é o uso da estrutura base de dados para manter os dados do sistema, entradas do modelo e resultados da simulação. O primeiro dá flexibilidade na definição do modelo e modificação enquanto o último permite rever resultados e a criação pelo usuário de vários tipos de

relatórios.

Tess apresenta oito características básicas que um sistema de simulação deve ter como forma de fornecer máxima utilidade [200]:

1. Suporta todos os aspectos de um projeto de simulação. O software reduz a quantidade de trabalho computacional necessário de forma que analistas com pouca habilidade computacional possam utilizá-lo de forma hábil e os de conhecimentos mais sofisticados possam utilizá-lo de forma mais eficiente.

2. Adequa os resultados para as necessidades do tomador de decisões. O tomador de decisões pode ver a simulação, assistir a sua validação e ganhar confiança em seus resultados, pois Tess anima o modelo e fornece representação gráfica de todos os resultados, com relatórios suplementares.

3. Aborda o projeto de simulação de forma modular. Seguindo estratégias de Engenharia de Software, Tess decompõe problemas grandes em séries de pequenos problemas ou módulos que são resolvidos a seu tempo. Fornece capacidades específicas para manejar cada módulo sendo construído, execução da simulação, análise dos resultados e apresentação, bem como um mecanismo de ligação destes módulos.

4. Minimiza a necessidade de conhecimento técnico computacional. Torna a operação da base de dados tão transparente quanto possível. Analistas podem executar todo o projeto sem referências diretas a base de dados. Fornece interfaces não programáveis para construir redes e controles, entrar dados e especificar a apresentação dos resultados.

5. Diferencia construtor de modelos de usuário de modelos. O usuário de modelos normalmente utiliza o modelo como uma caixa preta modificadora de entradas, sem se preocupar com detalhes internos. Contrutores querem detalhes da construção, das formas de entrada e saídas bem como a natureza dos mecanismos de apresentação. Tess suporta ambos.

6. Requer que o analista se concentre em somente uma forma do modelo. Utilizando características da linguagem de redes Slam, permite que o analista construa modelos de redes em um terminal gráfico interativo e atualize e modifique o modelo através do mesmo mecanismo. Tess elimina a necessidade de traduzir modelos da forma simbólica para forma declarativa.

7. Utiliza PCs como estações gráficas. Com sua função relatório pode criar arquivos seqüenciais de sua base de dados que pode ser descarregadas em um micro para a execução de tarefas de processamento. Contrariamente com softwares de micros, analistas podem criar arquivos de informações e repassá-los para o computador principal para uso na base de dados do Tess. Também utiliza um PC como um terminal gráfico.
8. Pode se adaptar a avanços de software e hardware. Independente de plataforma computacional, Tess pode adaptar-se a novos *drivers* que acompanham avanços na tecnologia gráfica e a novos computadores, especialmente os de 32-*bits*.

9.2.2. Casm

CASM (*Computer Aided Simulation Modelling*) é um conjunto de pesquisas que estão sendo desenvolvidos na Faculdade de Economia da Universidade de Londres, sobre formas de automatizar ao máximo a modelagem em simulação a partir do conceito de DCE [13]. A abordagem adotada é a das 3 fases que é a mais apropriada para trabalhar com DCE e a linguagem utilizada é o Pascal. Um conjunto de procedimentos e funções (pacote) para simulação codificado em Pascal foi construído e foi denominado eLSE (*extended Lancaster Simulation Environment*). Este pacote serviu de base para o Simul [165] que incorporou o conceito de amostragem descritiva [178].

Um programa gerador interativo para o eLSE foi construído e denominado Langen para auxiliar a formulação de problemas e foi desenvolvida uma interface para a linguagem natural NLUS (*Natural Language Understanding System*) com o objetivo de facilitar a comunicação homem-máquina dos modelos [115].

9.3. CONSIDERAÇÕES

Os softwares (programas geradores e pré-processadores) dão ao usuário os benefícios de uma entrada simbólica simples (especialmente valiosa para aqueles que não estão familiarizados com a prática da programação) combinada com o poder de uma linguagem de alto-nível.

Um programa gerador poupa o usuário das dificuldades

intrínsecas de uma linguagem de simulação geral. Embora permitindo somente um conjunto restrito de estruturas e conceitos, o programa gerador pode, muitas vezes, substituir inteiramente a linguagem principal para um grande subconjunto de problemas comuns.

Um ambiente suporte representa o estado-da-arte em simulação pois, é o que mais se aproxima da tarefa de automatizar um estudo completo de simulação, fornecendo apoio para todas as tarefas e não somente para a codificação do modelo.

CAPÍTULO X

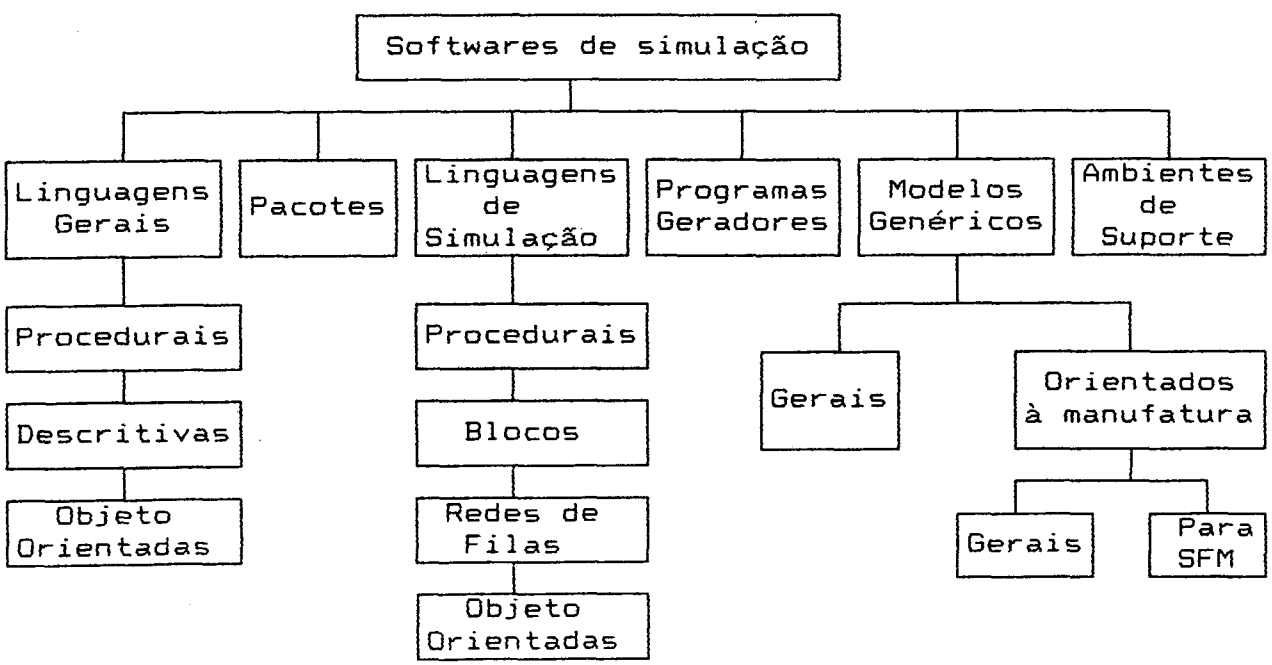
10. CONSIDERAÇÕES FINAIS

10.1. TENDÊNCIAS E PERSPECTIVAS DA SIMULAÇÃO DE SFM

Um modelo de simulação de um SFM assim como de qualquer outro sistema, pode ser realizada a vários níveis de esforço e com maior ou menor flexibilidade, sendo que quanto maior for a flexibilidade desejada maior também será o esforço de programação. Pode-se partir de um modelo totalmente livre, fazendo-se uso de uma linguagem de propósito geral, onde cada detalhe terá de ser construído até um modelo estilo 'camisa de força' dos pacotes orientados por dados, onde praticamente nenhuma programação deverá ser feita e somente os dados do problema deverão ser fornecidos.

A figura 10.1 apresenta um diagrama que resume as alternativas software, abordadas neste trabalho, de que se pode lançar mão para modelar um SFM. O nível de esforço computacional decresce da esquerda para a direita, assim como a flexibilidade.

Figura 10.1 - Taxionomia dos softwares para simular um sistema flexível de manufatura



Projetar um simulador não é diferente de projetar um sistema operacional. Ele pode ser conceituado por duas componentes principais: a base de dados e as regras. A base de dados deve ter flexibilidades embutidas que permitam fácil manipulação dos objetos da simulação como definição de novos objetos, adição atributos a objetos existentes, etc. Regras especificam como as ações devem ser tomadas, dependendo da conexão lógica entre objetos. Linguagens objeto orientadas são muito apropriadas, e por isso representarão importante papel no desenvolvimento futuro de softwares de simulação.

Num futuro próximo, mais simuladores serão escritos em linguagens objeto orientadas e lógicas, tais como, Prolog, Lisp, C++, Smalltalk, Pascal Objeto, etc. Estas linguagens tanto permitem a fácil manipulação dos objetos da simulação quanto fornecem interface direta com o desenvolvimento de sistemas especialistas (que utilizam as mesmas linguagens). Linguagens objeto orientadas serão usadas extensivamente na construção de futuros simuladores.

Qualquer experimento de simulação não-trivial, no passado, era realizado num mainframe, e além disso, todo software de simulação era baseado em computadores de grande porte. Os novos softwares de simulação são, quase na sua totalidade, baseados em microcomputadores ou, então, apresentam versões para estes equipamentos. O advento dos micros precipitou uma proliferação de pacotes de simulação baseados neste tipo de computador, com mais e mais características usuários-amigáveis. Tais características incluem: sistemas de dados de entrada dirigido por menu, representação avançada de relatórios e animação gráfica. A maioria das linguagens de simulação são ainda baseadas em Fortran. No entanto, na última década muitos softwares (principalmente pacotes) foram escritos em Pascal, C, Modula-2 e também em Ada.

Sistemas dirigidos por menu e animação gráfica são duas interfaces usuário-amigáveis disponíveis nas linguagens de simulação e nos pacotes modernos. A interface menu-dirigida (*menu-driven*) está se tornando de modo crescente uma interface padrão em micros. A interface tipicamente parte com um menu 'principal' listando opções para a seleção do usuário. Cada opção, ou guia o usuário a um sub-menu, ou chama uma execução numa opção selecionada. Já as linguagens de propósito geral estão evoluindo

para se tornarem verdadeiros sistemas de suporte a construção de softwares. Estruturas cada vez mais poderosas e flexíveis são fornecidas para a escrita, correção e manutenção de programas computacionais. Através da incorporação de editores de texto cada vez mais amigáveis, depuradores interativos e passo a passo, 'linkers', compiladores/interpretadores sempre mais eficientes e alguns casos macroprocessadores a tarefa de construção de programas longos (como os de simulação) fica cada vez mais facilitada.

Em termos de qualidade de animação, See-Why parece dar a impressão de tornar-se o padrão. Um recém-chegado é a linguagem de modelamento de sistemas Siman com seu animador Cinema. No entanto, os tipos de animação diferem uma vez que a do Siman não permite interação.

A tecnologia e implantação do SFM estão, atualmente, num rápido estágio de crescimento. Como tecnologia flexível de manufatura, avança em direção à maturidade, mais e mais metodologias para sua síntese e análise expandem-se, por esta razão, presencia-se uma proliferação de artigos na área.

Linguagens de simulação estão sendo gradualmente substituídas por simuladores especializados específicos num domínio de aplicação, tal como, projeto e análise de SFM. Esta tendência deverá se acelerar pelos recentes avanços no desenvolvimento de sistemas especialistas. O problema da entrada da coleção de dados, que é tipicamente grande consumidora de tempo, deverá ser substituída por um *'link'* direto para a base de dados do sistema real. Estes simuladores serão providos de interfaces convenientes com o usuário tal como animação gráfica e funcionalidades para análises posteriores.

10.2. PESQUISAS CORRENTES E PROGRESSOS

Os capítulos anteriores revisaram a diversidade de instrumentos de software disponíveis para simulação de sistemas dinâmicos a eventos discretos. Em um ambiente industrial o objetivo do usuário da simulação é melhorar a qualidade da tomada-de-decisões. Linguagens de alto-nível, programas geradores e softwares de modelamento genérico têm somente um objetivo parcial de ajudar a codificar o modelo e, desta forma, eles

naturalmente omitem caracteristicas importantes. Especificamente, eles não realizam adequadamente procedimentos experimentais diretos, não invocam os instrumentos estatísticos adequados e não fornecem saídas diversificadas apropriadas a variedade de usuários. Tais instrumentos encurtam o ciclo de codificação, eles não guiam o usuário ou reduzem a tarefa experimental. Em contraste, o levantamento ilustrou implementações que encaminham vários problemas.

A codificação de um modelo de simulação computacional é muitas vezes complicada pelas limitações da linguagem computacional usada para codificar o modelo. Linguagens de propósitos gerais não incluem estruturas de dados e funções necessárias em simulação. Como resposta a estas falhas, linguagens de simulação têm evoluído para assumir abordagens particulares de modelagem. Elas têm sido projetados para propósitos específicos e carentes de generalidade. Somente recentemente modeladores tem sentido a necessidade de um tratamento geral [235].

Ziegler [235], experimentou formalizar a modelagem baseando-a na teoria matemática dos sistemas. Entretanto, formalização tem pouco efeito na prática, sem meios adequados para implementação em um computador. Após quase três décadas de uso em simulação, os computadores ainda ofereciam pequena ajuda no desenvolvimento do modelo, na geração do programa de simulação e na verificação e validação do modelo. A falta de ferramentas adequadas de software e estruturas para organizar modelos e dados limitavam o uso de projetos modulares. Por sua vez, a falta de projetos modulares aumentava a dificuldade de melhorar e revisar modelos.

Existem duas abordagens principais para promover projetos modulares de modelos. Uma, é o desenvolvimento de linguagens baseadas em conceitos de sistemas teóricos, tais como Gest (*General System Theory Implementor*), que permite ao usuário manipular a estrutura do modelo e fornece facilidades para experimentação com modelos [208]. A segunda abordagem combina teoria dos sistemas e teoria dos autômatos. Teoria dos sistemas é usada para definir a estrutura e comportamento do modelo. Teoria dos autômatos é usada para analisar o comportamento do modelo [157].

Uma posição alternativa é recomendada por **Ziegler** [235].

Parte-se da representação abstrata do modelo, em direção a pragmática (prática). Embora não existam implementações comerciais, as idéias têm encontrado seguidores em vários centros. Proponentes da abordagem argumentam que uma linguagem formal, baseada em rigorosas ferramentas da lógica e da teoria dos conjuntos, proporcionará um método irrestrito e sem ambigüidades de formulação do modelo.

As comunidades acadêmicas e de pesquisa são capazes de vislumbrar, a longo prazo melhor do que o praticionista ou o consultor. Como consequência, existem vários programas de pesquisa que enfocam as necessidades diretas do modelador. Por exemplo, o programa JADE do Departamento de Ciência Computacional da Universidade de Calgary [220], que objetiva fornecer um conjunto integrado de instrumentos de software. Outro exemplo é o CASM (*Computer-Assisted Simulation Modelling*), cujo objetivo é fornecer ajuda para a formulação do problema, geração do programa e análise das saídas [13].

Exemplos adicionais são ROSS (*Rand Object-Oriented Simulation System*), sistema com base em IA, implementado em Lisp com o propósito de ser uma linguagem interativa e semelhante ao Inglês. Outro exemplo é o do Instituto Politécnico da Universidade Estadual da Virgínia que tem um longo interesse em sistemas de simulação e está correntemente engajado no estudo de ambientes de simulação. Os artigos publicados pelo grupo do Centro de Pesquisas da Virginia Sistemas, mostram que ela é uma das mais avançadas na área de desenvolvimento de sistemas integrados. Em seus artigos este grupo têm sugerido alguns novos aspectos e algumas novas soluções para a construção de modelos. Os objetivos são [147]:

1. Oferecer um suporte integrado e automático para o desenvolvimento do modelo do começo ao fim do seu ciclo de vida;
2. Melhorar a qualidade da tarefa de modelagem;
3. Aumentar significativamente a eficiência e a produtividade da equipe de projeto e
4. Diminuir substancialmente o tempo de desenvolvimento do modelo.

10.3. CONCLUSÃO

O simulador ideal para o SFM ainda não foi escrito. Tal

simulador deverá modelar adequadamente qualquer SFM concebível, rodar em qualquer computador disponível e ser usado por pessoas completamente não familiarizados com computação e simulação.

A simulação como ferramenta de análise de sistemas dinâmicos estocásticos, como os sistemas flexíveis de manufatura, não está livre de limitações. As mais citadas são: necessidade de treinamento especializado, longo tempo de modelamento, custo computacional alto e a inabilidade para otimizar. Contudo, com o avanço tecnológico dramático no hardware computacional e no software de simulação, estes obstáculos estão diminuindo. Linguagens de simulação orientadadas para a manufatura estão simplificando o processo de treinamento e encurtando o tempo de modelamento. Infelizmente, a inabilidade para otimizar não é alcançada por estes avanços.

Sistemas integrados de apoio à simulação, que fornecem ferramentas de software para a construção de modelos, construção de programas, gerenciamento de dados, construção de cenários, análise e apresentação de dados, estão também ajudando a este respeito. Microcomputadores, junto com linguagens de simulação para micros, estão reduzindo significativamente os custos das execuções. Embora análise de simulação seja ainda mais experimentação do que otimização, técnicas de modelamento híbridas estão ajudando a combinar as forças da simulação e da otimização.

Simulação é, normalmente, utilizada para tentar encontrar o 'valor ótimo' de variáveis decisórias de problemas que resistem a soluções analíticas. O objetivo é semelhante aos dos problemas de otimização e, desta forma, técnicas de programação matemática podem ser aplicadas à simulação, se bem que neste caso temos um complicador que é a natureza aleatória das respostas da simulação, requerendo tratamento estatístico. A literatura nesta área é esparsa. Uma abordagem compreensiva foi dada por **Safizadeh** [177], que identificou as técnicas de programação matemática, superfície de resposta, análise de perturbação, desenho experimental e domínio de freqüência com o objetivo de otimizar as saídas da simulação. Uma discussão da convergência da otimização em simulação é encontrado em **Curry** e **Hartfiel** [62].

Ficou claro que a simulação está bem estabelecida como a primeira ferramenta para estudar um SFM e seu uso está disseminado por todas as áreas. Modelos gerais e específicos estão sendo

desenvolvidos e utilizados por pesquisadores como instrumentos para aumentar a eficiência de sistemas existentes bem como para projetos de futuros SFMs.

Modelos generalizados são aqueles que tem muito da estrutura da tecnologia dos SFMs incorporado em seu código de simulação e o usuário especifica seu modelo particular através da entrada de dados. Modelos específicos são aqueles que são desenvolvidos para um único SFM, normalmente pelo próprio fabricante e não pode ser modificado através de dados para simular outros sistemas.

É interessante especular os motivos para o grande uso da simulação tanto por usuários quanto por vendedores para o estudo dos sistemas flexíveis de manufatura. Provavelmente a principal razão é que nenhuma outra ferramenta estava disponível para a tarefa. Necessidade é a mãe da invenção, neste caso, da introdução.

Influências secundárias na generalização da simulação são devidas a drástica redução de custos do hardware e do tempo de desenvolvimento dos modelos e como conseqüência da aceitação da simulação como ferramenta de resolução de problemas. Os custos de se utilizar simulação para resolver problemas tem sido dramaticamente reduzidos tanto pelo hardware cada vez mais eficiente quanto pelo software, isto é, pela melhoria nas técnicas de simulação. O progresso da tecnologia para o desenvolvimento de modelos não é menos dramático. Em 1960 modelos de simulação eram considerados exóticos e seu custo de desenvolvimento medido em homens/ano com duração da modelagem em meses. Hoje é possível desenvolver modelos de situações simples, através de menus, em 15 minutos e modelos razoavelmente complexos podem ser elaborados em horas. Um modelo generalizado de um SFM foi completado dentro de 3 semanas com um homem/semana de esforço [107].

No início dos anos 60 não era usual encontrar um curso de simulação em uma universidade americana. Por volta de 1980 o que não é comum é encontrar uma universidade que não tenha tal curso e muitas possuem vários cursos. Além disso a simulação era ensinada como um curso de computação em um departamento de ciência da computação. Hoje a simulação é vista como uma ferramenta de resolver problemas e é ensinada em qualquer departamento de Engenharia, Matemática Aplicada, Estatística e

Administração.

A tendência é clara, vendedores estão abastecendo seus consumidores com ferramentas de simulação. **Brian Moriarty** do laboratório *Draper* dos Estados Unidos coloca: "não é razoável para o usuário esperar por um vendedor para entender todos os assuntos envolvidos em sua situação particular. Desta forma, eles não podem depender somente dos vendedores para respostas rápidas e devem desenvolver seus próprios especialistas".

Duas áreas se destacam como críticas para a adoção da manufatura automatizada flexível. Para muitos, uma destas áreas é o custo. Posição esta que precisa ser revista, uma vez que investimento em SFM serão questão de sobrevivência econômica num futuro próximo, face ao competitivo mercado mundial. Outra área é a complexidade destes sistemas.

A máquina ferramenta convencional e seu operador constituem uma unidade autônoma, com um pequeno número de interfaces externas. O material é fornecido junto com novas ordens de serviço, pelo encarregado da produção. Possuindo um grande número de ferramentas padrão, o magazine de ferramentas só é visitado para substituição ou necessidades especiais. Este sistema operador/máquina é bastante autônomo e de complexidade limitada. A troca de componentes, operador ou máquina, não implica em alterações substanciais no sistema.

Os SFM, possuem um grande número de interfaces e o operador quando existe, desempenha a função de supervisor do processo. A unidade operador/máquina deixa de existir. Planejamento, controle, alimentação de materiais e ferramentas continuam a ser executados, só que total ou parcialmente automatizados. Desta forma, o processo produtivo deve ser pré-concebido com mais sofisticação. A troca de ferramentas evolui de uma simples entrega para um subsistema integrado ao sistema global de logística e comunicação da fábrica.

O uso de técnicas de simulação é uma decorrência lógica da complexidade do sistema, como forma de perceber, controlar, adequar, melhorar ou mesmo otimizar, até certo ponto, os vários subsistemas ou mesmo o sistema global. A organização de uma empresa nunca atinge uma condição final "ótima", já que restrições internas e externas mudam. As mudanças de mercado, novos materiais, novas tecnologias, novas formas de controle e

administração surgem constantemente, forçando um movimento contínuo de adaptações, melhoramentos, renovação, etc. A simulação entra como uma ferramenta para planejar, adequar, prever, organizar e 'otimizar' mudanças, como forma de evoluir continuamente, rumo a melhoria da qualidade e conseqüentemente da competitividade.

10.4. PESQUISAS ADICIONAIS

Uma vez que um modelo de simulação é na verdade, em parte, um programa computacional, avanços na área de software e hardware constituem na realidade avanços em simulação. Estes avanços se estendem em todas as direções e ocorrem continuamente tornando um levantamento completo impossível, assim este trabalho ficou limitado a simulação tradicional. Desta forma, tópicos de ponta como simulação concorrente ou paralela e simulação inteligente não foram abordados. A área de software é muito ampla e dinâmica, de modo que, seria impossível querer abrangê-la por completo, mesmo por que, neste meio tempo muitas novidades terão surgido, como por exemplo, pacotes e linguagens objeto orientadas, assim como, os conceitos de modelagem e simulação visual interativa. Possivelmente, a maioria das linguagens, pacotes e outros softwares abordados podem resultar em trabalhos de dissertação. Um trabalho interessante que parece não ter sido ainda realizado é uma comparação entre as várias linguagens de programação de propósito geral, com respeito às suas potencialidades na codificação de um mesmo modelo de simulação. Uma comparação entre uma linguagem tradicional e uma objeto orientada para construir um modelo de simulação seria um trabalho interessante, pois parece que este tipo de linguagem está indicando a tendência futura para o desenvolvimento de softwares.

Outro trabalho que podem ser realizado como complemento, e que inclusive já estou encaminhando, é sobre as demais etapas do estudo de simulação como a modelagem da estática do sistema (DCE e redes de filas), a modelagem da dinâmica do sistema (abordagens do evento, atividade e processo).

A modelagem da aleatoriedade do sistema, isto é, identificação das distribuições de probabilidade apropriadas, a geração de números aleatórios e das distribuições de probabilidade

também é um trabalho que pode dar continuidade a este.

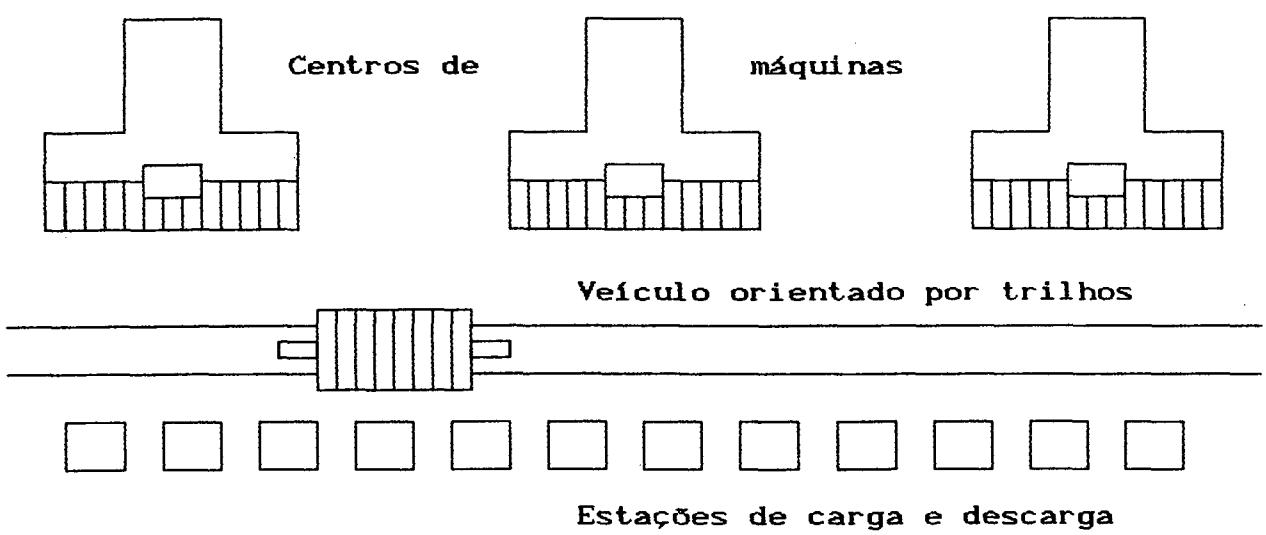
A análise dos resultados da simulação encontra-se, hoje, desenvolvida o suficiente para, também, merecer um trabalho específico. O estudo dos tipos de simulação: finalizada e estado-estacionário, bem como das técnicas estatísticas para estimar as variáveis resposta no caso de uma simulação com propósitos preditivos e análise fatorial, bem como, das técnicas de otimização desta análise, se a simulação tiver propósito comparativo.

APÊNDICE

EXEMPLO

Este exemplo ilustra a modelagem de um SFM, utilizando um tipo de representação gráfica, denominada de diagrama ciclo da entidade (DCE), para um SFM simples, muito parecido com o SFM de torneamento vertical da Caterpillar. O sistema consiste de três máquinas, um veículo e doze estações de carga, conforme figura A.1. A posterior codificação do modelo poderá ser feita diretamente por um dos muitos pacotes que aceitam este tipo de representação como entrada, como por exemplo, o Hocus, ou ainda, através dos programas geradores Caps ou Draft.

Figura A.1 - SFM da Caterpillar (EUA)



As estações de carga são dedicadas a um tipo particular de peça e também servem como local de armazenagem de pallets.. Algumas hipóteses simplificadoras são feitas. Enquanto no sistema real muitas das peças necessitam de duas ou mais operações e alguns fixadores são usados por dois tipos de componentes, aqui assume-se que cada peça requer somente uma operação e cada pallet é usado por somente um tipo de peça. O veículo no sistema real tem

duas posições de carga, mas, neste exemplo, assume-se que somente um pallet será transportado a cada vez. Desta forma, as principais hipóteses são:

1. O veículo tem somente uma posição de carga.
2. Cada peça requer somente uma operação.
3. Operações podem ser executadas em qualquer máquina.
4. As máquinas não apresentam falhas.
5. Cada estação de carga serve somente a um tipo de peça e pallet.
6. Os fixadores estão permanentemente montados nos pallets.

As entradas do modelo serão:

Máquinas, componentes, pallets, estações de carga e descarga, operadores e veículos.

Uma vez que existe um número igual de estações de carga/descarga e pallets e cada estação de carga é dedicada a um pallet particular, a disponibilidade de estações de carga não impede a ocorrência das atividades. Desta forma pode-se omitir as estações de carga e descarga do modelo.

O diagrama ciclo da entidade (DCE) representa o ciclo de vida de cada tipo de entidade do sistema. Estas entidades são: operador, componentes (peças), veículos, máquinas e pallets. Cada uma destas entidades tem um ciclo de vida fechado constituído de atividades (ações) e filas (esperas). O diagrama é construído com base nas seguintes convenções:

1. Cada tipo de entidade tem uma atividade.
2. O ciclo consiste de atividades e filas.
3. Atividades e filas se alternam no ciclo.
4. O ciclo é fechado e
5. Atividades são representadas por retângulos e filas por elipses. O ambiente (fila infinita) por duas elipses superpostas.

CONVENÇÕES

O diagrama da figura A.2, utiliza as seguintes convenções:

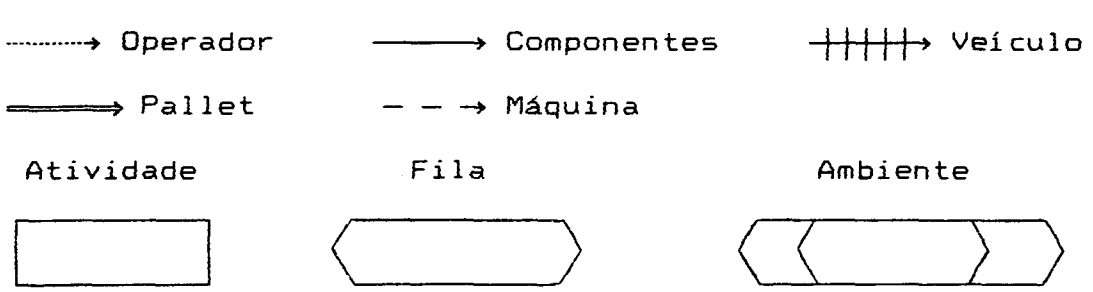
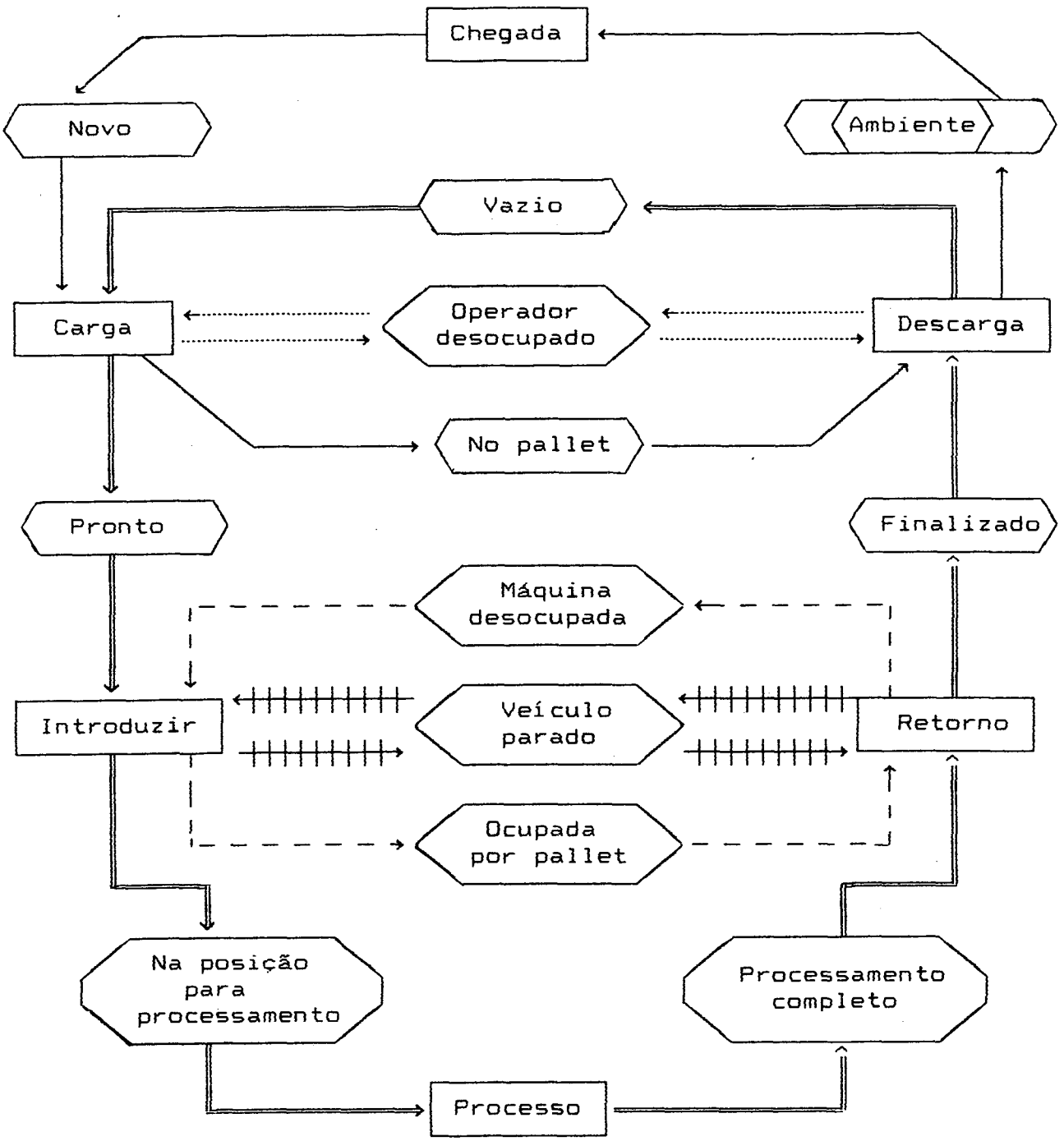


Figura A.2 - DCE para o SFM da Caterpillar (EUA)



GLOSSÁRIO

Acurácia. Qualidade, estado ou grau de conformação a um padrão estabelecido. Diferença entre a resposta real e a desejada ou comandada em um sistema de controle automático.

Algoritmo. Seqüência de passos exigidos para se resolver um problema.

AI (*Artificial Inteligence*). O mesmo que inteligência artificial.

AGV (*Automated Guided Veicle*). Veículo dirigido automaticamente. Um dos meios de movimentação de materiais dentro de uma fábrica.

AS/RS. Abreviatura de *Automated Storage/Retrieval System* ou Sistema de armazenagem e recuperação automáticos.

Bit (*BInary digiT*). É a menor unidade de informação computacional.

Byte. Palavra básica computacional. Grupo de oito bits.

CAD (*Computer-Aided Design*). Projeto auxiliado por computador.

CAE (*Computer-Aided Engineering*). Engenharia auxiliada por computador.

CAM (*Computer Aided Manufacturing*). Fabricação assistida por computador.

C/D. Abreviatura de Carga e Descarga.

Centro de usinagem. O mesmo que *machining center*.

CIM (*Computer Integrated Manufacturing*). Manufatura integrada por computador.

CLP. Controlador lógico programável. O mesmo que PLC.

CN. Comando Numérico. O uso de informações numéricas codificadas ao invés de um operador para controlar uma máquina ferramenta.

CNC (*Computer Numerical Control*). Um sistema de comando numérico em que um computador dedicado é usado para armazenar um programa de controle da máquina.

CND. Um sistema conectando um conjunto de máquinas NC para uma memória comum de armazenamento de programas de peças. O suprimento é feito por controle *on line*. Geralmente elimina a fita.

Concorrente. Execução de um processo dentro de um mesmo intervalo de tempo, enquanto se realiza outros. O mesmo que processamento paralelo.

Configuração. Uma combinação particular de um computador, softwares e módulos de hardware e periféricos em uma única instalação e interconectados de forma a suportar certas aplicações.

Controle. O processo de fazer uma variável ou sistema de variáveis comportar-se como o desejado.

Controle adaptativo. Um método de controle em que parâmetros de controle são continuamente e automaticamente ajustados em resposta a variáveis de processo medidas para obter um desempenho quase-ótimo.

Controle analógico. Controle envolvendo processamento de sinais de aparelhos analógicos (Eletrônicos, hidráulicos, pneumáticos, etc.).

CPU (*Central Processing Unit*). Unidade central de processamento.

Data-drive. Designação de um software que praticamente não requer programação, apenas a entrada dos dados.

Dedicado. Aparelho projetado para uma aplicação específica.

DNC (*Direct Numerical Control*). O mesmo que CND.

2-D. Utilizado em relação a softwares que apresentação animação em duas dimensões, ou seja, apresentação de figuras geométricas planas.

Emular. O mesmo que imitar.

ES (*Expert System*). O mesmo que sistema especialista.

Facilities. Palavra inglesa para designar os equipamentos que viabilizam a manufatura. Normalmente significa as instalações.

Fixador (*Fixture*). Mecanismo utilizado para posicionar e fixar uma peça nas máquinas ferramentas.

FMS (*Flexible Manufacturing System*). O mesmo que sistema flexível de manufatura.

Hardware. Designação geral do computador (máquina) e seus componentes físicos. Palavra inglesa que significa 'ferragens', mas propositadamente utilizada para representar os aspectos tecnológicos físicos do computador em oposição aos operacionais (linguagens e programas).

Heurística. Método de resolução de problemas por tentativa e erro ou também em as soluções são descobertas por avaliação do progresso feito em direção ao resultado final.

Host computer ou computador *host.* O principal computador em um sistema computacional ou rede. Ele executa serviços tais como gerenciamento da base de dados que outros computadores e processadores na rede não estão configurados para executar com eficiência. No texto é utilizado o termo computador principal.

IA . Inteligência Artificial, qualidade atribuída a programas e

computadores que dão a ilusão de uma atividade intelectual humana. Uso de computadores para simular processos inteligentes.

Indexador. Movimentador dos acessórios de uma máquina de modo que uma operação seja repetida a certos intervalos de tempo.

Input. O *mesmo* que entrada. Opõem-se a *output* ou saída.

Interativo. Significa a comunicação dupla entre o software e o usuário. O usuário recebe um *feedback* do sistema para orientação e verificação.

Interface. Uma ligação de hardware ou software que permite a dois sistemas ou a um sistema e seus periféricos operarem como um único sistema integrado.

I/O. Forma abreviada de *Input/Output*, i. é entrada/saída.

Job Shop. Palavra inglesa para designar uma empresa de manufatura em pequenos lotes, dedicada a manufaturar peças ou produtos especiais para consumidores específicos e geralmente sob encomenda.

LAN (*Local Area Network*). Rede local.

Layout. O arranjo físico dos ítens de produção dentro da fábrica. Incluindo escritórios, departamentos, almoxarifados e equipamentos de produção.

Lead Time. Tempo que um produto leva para ser completado em um processamento normal

Linha de transferência (*transfer line*). Máquinas para trabalhar metais em alto volume para ambientes tais como indústria automotiva que pode tolerar inflexibilidade.

LPG. Linguagem de Programação Geral que não está voltada a uma área específica.

LS. Linguagem de Simulação. São aquelas projetadas especialmente para simulação. Engloba as gerais e as especiais (de manufatura).

L/U. Abreviatura de *Loading and Unloading.* O mesmo que carga e descarga, C/D.

Mainframe. Um computador central, capaz de executar um grande número de cálculos, gerenciar base de dados e suportar grande número de terminais. Normalmente tem palavra de 32 ou mais *bits.*

Manufatura. A transformação da matéria prima em produto final.

Map (*Manufacturing Automation Protocol*). Protocolo automático da manufatura. Conjunto de padrões estabelecidos pela GM para automatizar a manufatura.

Máquina ferramenta. Máquina utilizada para cortar ou moldar

metais.

Machining center (Centro de usinagem). Máquina capaz de executar uma variedade de operações de remoção de metal em uma peça, normalmente sob controle numérico.

Menu. Uma lista de opões disponíveis ao usuário de um software computacional.

MM. Abreviatura de movimentação de materiais.

Microprocessador. Um *chip* com capacidade de processamento, mas sem memória aleatória (ou dispositivos de entrada e saída). Pode servir como uma CPU de um micro.

Minicomputador. Um tipo de computador em que os elementos básicos da CPU são constituídos de vários componentes discretos e circuitos integrados ao invés de um único circuito integrado como num microcomputador.

Mix. Conjunto de produtos fabricados por uma empresa ou também conjunto de produtos processados em um mesmo lote.

Pallet. Uma plataforma em que peças são colocadas e que podem ser transportadas.

Peças (*Part* ou *workpiece*). Uma componente física, não uma montagem de outros ítens.

Periféricos. Equipamentos que funcionam em conjunto com um sistema de comunicação ou um computador, mas não faz parte dele. São as impressoras, terminais, *ploters*, etc.

PLC (*Programable Logic Controller*). Um microprocessador que pode ser programado para controlar equipamentos industriais. No texto utiliza-se o termo CLP.

PNB. Produto Nacional Bruto.

Protocolo. Um conjunto de regras dirigindo o uso de um sistema computacional ou a transmisão de dados entre aparelhos e sistemas.

NC (*Numerical Control*). O mesmo que comando numérico.

Output. Saída. Opõem-se a *Input* (Entrada).

Recursos. Dinheiro disponível para a operação da empresa. Mais direitos como: patentes, contratos ou privilégios. Também materiais disponíveis através do uso do dinheiro tal como matéria prima.

RS-232. Uma interface padrão para unir computadores e periféricos.

SE. Sistema Especialista. Programa ou conjunto de programas projetados para trabalhar com problemas altamente especializados, no âmbito da IA. Dispõe de uma base de conhecimentos numa área

restrita ou domínio particular.

SFM. Abreviatura de Sistema Flexível de Manufatura.

Set up ou setup. Tempo de preparação para a execução de uma operação. *Set up* de máquina envolve equipar a máquina com os acessórios apropriados, ferramentas, fixadores e estabelecer parâmetros como velocidade e profundidade de corte, etc.

Shutle. Palavra inglesa para designar um veículo que se movimenta regularmente entre dois pontos.

Software. Conjunto de instruções de operação fornecidos a um computador. Sinônimo de programa ou linguagem de programação. Opõe-se a hardware.

Tempo real. Tradução de *real time*, ou seja, processamento que ocorre simultaneamente com a entrada dos dados. É o mesmo que *on line*, ou seja, que está ligado diretamente na linha de processamento.

Throughput time. Tempo de fabricação. Tempo gasto para completar um lote de peças ou montagens, iniciando no momento em que os materiais estão prontos para a primeira operação e terminando com a conclusão da última operação em todo o lote.

Top-down. De cima para baixo.

3-D. Significa um software com animação em terceira dimensão, ou seja com figuras geométricas espaciais.

Usuário-amigável (*User-friendly*). A característica de um programa que descreve sua facilidade de uso.

WIP (*Work-In-Process* ou *Work-In-Progress*). Produtos em vários estágios de acabamento através do ciclo de produção, incluindo matéria prima que foi liberada para o processamento inicial e produtos acabados aguardando inspeção final. O mesmo que trabalho em processo ou em progresso. No texto é utilizado o termo produto em processo.

REFERÊNCIAS

[001] ABED, Seraj Yousef. A Quantitative Comparison of Three Simulation Languages: SLAM, GPSS/H, SIMSCRIPT. *Computers & Industrial Engineering*, USA, v. 9, n. 1, p. 45-66, 1985.

[002] ABDIN, M. F., MOHAMED, N. S. The Role of Simulation in Design of FMSs. *Proceedings of the 8th Annual Conference on Computers and Industrial Engineering*, p. 372-76.

[003] ADELSBERGER, Heimo H. Prolog as a Simulation Language *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 501-04, 1984.

[004] ADKINS, Gerald, POOCH, Udo W. Computer simulation: A tutorial. v. 10, n. 4, p. 12-17, 1978.

[005] ALBERTI, N., DIEGA, Noto La, PASSANNANTI, A. Cost Analysis of FMS Throughput. *Annals of the CIRP.* Germany, v. 37, n. 1, p. 413-416, 1988.

[006] ALMODOVAR, Angel R. Manufacturing Simulators Save Time, Provide Good Data for Layout Evaluation. *Industrial Engineering.* USA, v. 20, n. 6, p. 28-33, June 1988.

[007] ALTING, Leo, BILBERG, Arne, LARSEN, Niels Erik. Extended Applications of Simulation in Manufacturing Systems. *Annals of the CIRP*, Germany, v. 37, n. 1, p. 417-20, 1988.

[008] ANG, Alfredo H-S., TANG, Wilson H. *Probability Concepts in Engineering Planning and Design.* Volume II: Decision, Risk, and Reliability. New York: John Wiley & Sons.

[009] ARGENTA, Ilson Luiz. Um Método de Modelagem de Sistemas Discretos de Manufatura com Redes de Petri de Alto Nível. In: *XX Congresso Nacional de Informática*, 1987, São Paulo, *Anais...* p. 615-23.

[010] ASFAHL, C. Ray, BALAGAMWALA, Ashfaq. Simlog: A Prolog-Based Simulator for Industrial Logic Control Systems. *Computers Industrial Ingineering*, Great Britain, v. 19, n. 1/4, p. 195-99, 1990.

[011] AVONTS, Ludwig H., GELDERS, Ludo F., WASSENHOVE, Luk N. Van. Allocating Work Between an FMS and a Conventional Jobshop: A Case Study. *European Journal of Operational Research.* North-Holland, v. 33, n. 3, p. 245-56, Feb 1988.

[012] AZZONE, Giovanni, BERTELE, Umberto. Measuring the Economic Effectiveness of Flexible Automation: A New Approach.

International Journal of Production Research, London, v. 27, n. 5, p. 735-746, May 1989.

[013] BALMER, David W., PAUL, Ray J. CASM-The Right Environment for Simulation. *Journal of Operational Research Society*, Great Britain, v. 37, n. 5, p. 443-52, May 1986.

[014] BANKS, Jerry, CARSON II, John S. *Discret-Event system simulation*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc,. 1984, 514 p.

[015] BANKS, Jerry, CARSON II, John S. Process-Interaction Simulation Languages. *Simulation*, USA, v. 44, n. 5, p. 225-35, May 1985.

[016] BASTOS, José M. Batching and Routing: Two Functions in the Operational Planning of Flexible Manufacturing Systems. *European Journal of Operational Research*, North-Holland, v. 33, n. 3, p. 230-44, Jan. 1988.

[017] BEASLEY, J. E. Supercomputers and OR. *Journal of Operational Research Society*. Great Britain, v. 38, n. 11, p. 1085-89, Nov. 1987.

[018] BEEK, Paul, PUTTEN, Cornelis van. OR Contribuitions in Flexibility and Improvement in Production/Inventory Systems. *European Journal of Operational Research*, North-Holand, v. 31, n. 1, p. 52-60, July 1987.

[019] BELL, Peter C. Visual Interactive Modelling in Operational Research: Successes and Opportunities. *Journal of Operational Research Society*, Great Britain, v. 36, n. 11, p. 975-82, Nov. 1985.

[020] BEN-ARIEH, D. Manufacturing System Application of a Knowledge Based Simulation. *Proceedings of the 8th Anual Conference on Computers and Industrial Engineering*. 459-63

[021] BENGTSON, Neal M. Microcomputer vs. Mainframe Simulations: A Case Study. *Software-Practice and Experience*, England, v. 19, n. 10, p. 957-65, Oct. 1989.

[022] BETTONVIL, Bert. A Formal Description of Discrete Event Dynamic Systems Including Infinitesimal Pertubation Analysis. *European Journal of Operational Research*, North-Holland, v. 42, n. 2, p. 213-22, Sept. 1989.

[023] BEVANS, Patrick John. First, Choose an FMS Simulator. *American Machinist*, USA, v. 126, n. 5. p. 143-45, May 1982

[024] BONNEY, M. et al. The Simulation of Industrial Robot Systems. *OMEGA International Journal of Management*

Science, Great Britain, v. 12, n. 3, p. 273-281, May/June 1984.

[025] BONETTO, Roger. *Flexible Manufacturing Systems in Practice*. London: North Oxford Academic, 1988, 208 p.

[026] BOSE, Partha Protim. Simulation Models Factory Floors. *American Machinist*, USA, v. 132, n. 8, p. 49-53, Aug. 1988.

[027] BRANSTRATOR, John, McCORMICK, Melissa. Computer Simulation Justifies Construction of CIBA-GEIGY Dyestuff Production Facility. *Industrial Engineering*, USA, v. 21, n. 5, p. 17-20, May 1989.

[028] BRILL, P. H., MANDELBAUM, M. Measurement of adaptivity and flexibility in prodution systems. *European Journal of Operational Research*, North-Holland, v. 49, n. 3, p. 325-32, Dec. 1990.

[029] BRILL, P. H., MANDELBAUM, M. On Measures of Flexibility in Manufacturing Systems. *International Journal of Production Research*, London, v. 27, n. 5, p. 747-56, Sept./Oct. 1989.

[030] BRINKMAN, Bas. Strategy for Flexibility. *Proceedings of the Second IFIP WG 5.7 Working Conference on Advancing Production Management Systems-APMS 85*, Hungary, p. 241-48, Aug. 1985.

[031] BROWN, Evelyn. IBM Combines Rapid Modeling Technique and Simulation to Design PCB Factory-Of-The-Future. *Industrial Engineering*, USA, v. 20, n. 6, p. 23-26,90, June 1988.

[032] BRUNO, Giorgio, CONTERNO, Renato, MINERO, Valerio. Hybrid Simulation of a Flexible Manufacturing System. *Proceedings of the Second IFIP WG 5.7 Working Conference on Advances in Production Management Systems-APMS 85*, Hungary, p. 131-444, Aug. 1985.

[033] BRYANT, Raymond M. Discrete system simulation in Ada. *Simulation*, USA, v. 39, n. 4, p. 111-21, Oct. 1982.

[034] BULLERS, W. I et al. Artificial Inteligence in Manufacturing Planning and Control. *AIIE Transactions*. USA, v. 12, n. 4, p. 351-63, 1980.

[035] BUZACOTT, J. A., YAO, David D. Flexible Manufacturing Systems: A Review of Analytical Models. *Management Science*, USA, v. 32, n. 7, p. 890-905, July 1986.

[036] CAMBRON, Kenneth E., EVANS, Gerald W. Layout Design Using The Analytic Hierarchy Process. *Computers in Industrial*

Engineering, USA, v. 20, n. 2, p. 211-29, 1991.

[037] CAMP, Robert E. Simulation Determines Storage Requirements at Reynolds Tobacco Company. *Industrial Engineering*, USA, v. 22, n. 1, p. 28-31, Jan. 1990.

[038] CAMP, R. E. Storage Requirements are Determined through the Use of Simulation. *Industrial Engineering*, USA, v. 22, n. 3, p. 44-6, Mar. 1990.

[039] CARRIE, Allan. *Simulation of Manufacturing Systens.* Great Britain: John Wiley & Sons, 1988, 417 p.

[040] CARRIE, Allan S. The Role of Simulation in FMS. *Flexible Manufacturing Systems: Methods and Studies. Edited by Andrew Kusiak*, North-Holland, v. 12, p. 191-208, 1986.

[041] CARRIE, A. S. et al. Introducing a Flexible Manufacturing system. *International Journal of Production Research*, London, v. 22, n. 6, p. 907-16, Nov./Dec. 1984.

[042] CARVALHO, R. Spinelli de, CROOKES, John G. Cellular Simulation. *Operational Research Quarterly*, London, v. 27, n. 1/i, p. 31-40, 1976.

[043] CARVALHO, Wilson Marques. CIM - Introdução da Manufatura Celular. In: *7. SCNB - Seminário de Comando Numérico no Brasil e 3. Jornada Internacional de Automatização Industrial*, 1987, São Paulo. *Anais...* p. 01.01-01.36.

[044] CAULLIRAUX, Heitor Mansur, NAVEIRO, Ricardo Manfredi. Projeto e Simulação de Células de Produção com Uso de Redes de Petri. In: *9. SCNB - Seminário de Comando Numérico no Brasil e 5. JIAI - Jornada Internacional de Automatização Industrial*, 1989, COPPE/UFRJ. *Anais...* p. 05.01-05.19.

[045] CELLIER, F. E. *Progress in Modelling and Simulation.* USA: Academic Press, 1982, 466 p.

[046] CHANG, YIH-LONG, SULLIVAN, Robert S, WILSON, James R. Using SLAM to design the material handling system of a flexible manufacturing system. *International Journal of Production Research*, UK, v. 24, n. 1, p. 15-26, 1986.

[047] CHEN, Kan. Generation Dynamics in Computer-Integrated Manufacturing Systems. *Techonological Forecasting and Social Change*, North-Holland, v. 31, n. 1, p. 19-26, Mar 1987.

[048] CHEN, Shin-Ken. *The Design, Evaluation and Test of Flexible Manufacturing System : A hybrid Approach of Optimization*

and Phisical emulation. USA: Case Western Reserve University, May 1987. Ph.D. Thesis.

[049] CHENG, T. C. E. Simulation of Flexible Manufacturing Systems. *Simulation*, USA, v. 45, n. 6, p. 299-302, Dec. 1985.

[050] CHISMAN, James A. Apple Uses Simulation to Improve PCB/FMS Line Design and Operation. *Industrial Engineering*, USA, v. 21, n. 7, p. 40-41, July 1989.

[051] CHOW, We-Min, MACNAIR, Edward A., SAUER, Charles H. Analysis of Manufacturing Systems by the Research Queueing Package. *IBM Journal of Research and Development.* v. 29, n. 4, p. 330-42, July 1985.

[052] CHRISTY, David P., WATSON, Hugh J. The Application of Simulation: A Survey of Industry Practice. *Interfaces*, USA, v. 13, n. 5, p. 47-52, Oct 1983.

[053] CHU, C. C., TALAVAGE, J. J. Simulation Optimization for Decision Support in Operating a Robotic Manufacturing System. *Proceeding of the 1984 Winter Simulation Conference.* 221-24, 1984.

[054] CLEMENTSON, A. T. Extended Control and Simulation language. *The Computer Journal*, London, v. 9, n. 2, p. 215-20, 1966.

[055] CO, Henry C., CHEN, S. K. Design of a Model Generator for Simulation in Slam. *Engineering Cost and Production Economics*, Amsterdam, v. 14, p. 189-98, 1988.

[056] CO, Henry C., WYSK, Richard A. The robustness of CAN-Q in modelling automated manufacturing systems. *International Journal of Production Research*, London, v. 24, n. 6, p. 1485-503, 1986.

[057] CO, Henry, LIU, Jianhua. Simulation and Decision Analysis in FMS Justification. *Proceeding of the 1984 Winter Simulation Conference*, USA, p. 407-12, 1984.

[058] COHEN, P. H., WYSK, R. A. Digital and Physical Simulation of Manufacturing Systems. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 381-83, 1984.

[059] COX, Springer. The User Interface of GPSS/PC. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 545-550, 1984.

[060] CROOKES, John G. et al. A Three-Phase Simulation System Written in Pascal. *Journal of Operational Research Society*, Great Britain, v. 37, n. 6, p. 603-18, June 1986.

[061] CROOKES, John G. Generators, Generic Models and Methodology. *Journal Of Operational Research Society*. Great Britain, v. 38, n. 8, p. 765-68, Aug. 1987.

[062] CURRY, Guy L., HARTFIEL, D. J. A Simulation-Optimization Method: Its Convergence and Utility. *Naval Research Logistics Quarterly*, New York, v. 30, n. 2, p. 227-36, 1983.

[063] DACHS, J. Norberto W. *Estatística Computacional: Uma introdução em Turbo Pascal*. São Paulo: Livros Técnicos e Científicos Editora Ltda. 1988, 236 p.

[064] DAVIES, Ruth M. Davies, O'KEEFE, Robert M. *Simulation Modelling with Pascal*. UK: Prentice-Hall Ltd, 1989, 303 p.

[065] DIESCH, Kurt H., MALSTROM, Eric M. Physical Simulator Analyzes Performance of Flexible Manufacturing System. *Industrial Engineering*, USA, v. 17, n. 6, p. 66-77, June 1985.

[066] DENING, Peter J., BUZEN, Jeffrey P. The operational analysis of queueing network models. *Computing Surveys*, New York, v. 10, n. 3, p. 225-61, Sept. 1978.

[067] DUDEWICZ, Edward J., KARIAN, ZAVEN A (Editores). *Tutorial: Modern Design and Analysis of Discrete-event Computer Simulations*. IEEE Computer Society, 1985, 475 p.

[068] DWYER John & KORVIN, Steve. Honeywell Puts Simulation To Work In Multiple Arenas. *Industrial Engineering*, USA, v. 22, n. 10, p. 33-36, Oct. 1990.

[069] EKERE N. N., HANNAM, R. G. An Evaluation of Approaches to Modelling and Simulating Manufacturing Systems. *International Journal of Production Research*, London, v. 27, n. 4, p. 599-611, Apr. 1989.

[070] ECKER, Joseph G, KUPFERSCHMID, Michael. *Introduction to Operations Research*. New York: John Wiley & Sons, 1988.

[071] EL-RAYAH, T. E., HOLLIER, R. H. A Review of Plant Design Techniques. *The International Journal of Production Research*, England, v. 8, n. 3, p. 263-79, Oct. 1970.

[072] EVERSHEIM, Walter. Conceitos Modernos de Manufatura na Era CIM. *Boletim Sobracon*, São Paulo, Ano vii, n. 47, p. 75-93, nov./dez. 1989 e jan./fev. 1990.

[073] FALKNER, Charles H., SHANKER, Natraj. Physical Simulation of Flexible Manufacturing Systems. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 397-404.

[074] FAHMY, Ahmed Fikry Abdel Wahab. *An Expert System for the Design of Flexible Manufacturing Sistems Using Simulation Analysis.* USA: University of Alabama, 1986. Ph.D. Thesis.

[075] FERREIRA, Eduardo Damasceno, PASKULIN, Flávio Alberto. AGVS (Sistemas Autônomos de Transporte), uma Opção para a Racionalização do Fluxo de Material e seu Potencial Aplicativo. *9º SCNB - Seminário de Comando Numérico e Automatização Industrial e 5º JIAI - Jornada Internacional de Automatização Industrial*, 1989, São Paulo, *Anais* ... p. 16.1-16.24.

[076] FORD, D. R., SCHROER, Bernard J. An Expert Manufacturing Simulation System. *Simulation.* USA, v. 48, n. 5, p. 193--200, May 1987.

[077] FORTIER, Paul J. Design and Analysis of Distributed Real Time Systems. New York: Intertext Publications, Inc., McGraw-Hill, Inc., 1985

[078] FRIEL, Patricia, SHEPPARD, Sallie. Implications of the Ada Enviroment for Simulation Studies. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 477-89, 1984.

[079] FRY, Timothy D., SMITH, A. E. FMS Implementation Procedure: A Case Study. *IEE Transactions*, USA, v. 21, n. 3, p. 288-93, Sept. 1989.

[080] GUEZZI, Carlo, JAZAYERI, M. *Programming Language Concepts.* United States: John Wiley & Sons, 1987, 426 p.

[081] GHOSH, Biman K. Design and Performance Analysis Models of Computer Networks in CIM Systems. *Computers in Industry*, Amsterdan, v. 12, n. 2, p. 141-52, May 1989.

[082] GHOSH, Biman K., WYSK, Richard A. Models for Communication System Performance in Computer Integrated Manufacturing Systems. *Computers & Industrial Engineering*, Great Britain, v. 16, n. 1, p. 55-64, 1989.

[083] GHOSH, Biman K., WYSK, Richard A. Post-Optimality Analysis of Performance of Computer Networks in Flexible Manufacturing Systems. *Computers & Industrial Engineering*, Great Britain, v. 16, n. 1, p. 45-53, 1989.

[084] GILBERT, James P., WINTER, Peter J. Flexible Manufacturing Systems: Technology and Advantages. *Production and Inventory Management*, USA, v. 27, n. 4, p. 53-9, Fourth Quarter, 1986.

[085] GOGG, Thomas, SANDS, Charles D. Hughes Aircraft Designs

Automated Storeroom Systems Through Simulation Application. *Industrial Engineering*, USA, v. 22. n. 8, p . 49-57, Aug. 1990.

[086] GOOD, George L., BAUNER, J. Thomas. On the Use of Simulation in the Design and Installation of a Power and Free Conveyor System. *Proceeding of the 1984 Winter Simulation Conference*, USA, 1984.

[087] GORDON, Geoffrey. *System Simulation*. Englewood Cliffs, New York: Prentice-Hall, Inc. 2nd ed. 1978.

[088] GRANT, Floyd H., MacFARLAND, Douglas G. Simulation With C. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 491-95, 1984.

[089] GREEN, Richard G. Flexible manufacturing systems: Are they in your future? *Tooling & Production*, USA, v. 52, n. 4, p. 35-38, July 1986.

[090] GREENWOOD, Nigel R. *Implementing Flexible Manufacturing Systems*. New York: John Wiley & Sons, 1988, 279 p.

[091] GROOVER, Mikell P. *Automation, Production Systems, and Computer-Aided Manufacturing*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1980.

[092] GUPTA, Yash P., GOYAL, Sameer. Flexibility of Manufacturing Systems: Concepts and Measurements. *European Journal of Operational Research*. North-Holland, v. 43, n. 2, p. 119-35 Nov. 1989.

[093] HAC, Anna. Computer System Simulation in Pascal. *Software--Practice and Experience*, USA, n. 12, p. 777-84, 1982.

[094] HADDOCK, Jorge. A Simulation Generator for Flexible Manufacturing Systems Design and Control. *IIE Transactions*. U.S.A., v. 20, n. 1, p. 22-31, Mar 1988.

[095] HARMONOSKI, Catherine M., SADOWSKI, Randall P. A Simulation Model and Analysis: Integrating AGV'S with Non-Automated Material Handling. *Proceeding of the 1984 Winter Simulation Conference*, USA, p. 341-47, 1984.

[096] HEGLAND, Donald E. Flexible Manufacturing - Your Balance Between Productivity and Adaptability. *Flexible Manufacturing Systems. Edited by John R. Holland and published by SME*. p. 169-86, 1984.

[097] HENRIKSEN, James O. Discrete Event Simulation Languages Current Status and Future Directions. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 83-88, 1984.

[098] HIGDON, Jim. Planning a New Material Handling System. *Industrial Engineering*, USA, v. 20, n. 11, p. 55-59, Nov. 1988.

[099] HILLIER, F. S., LIEBERMAN, G. J. *Introdução a Pesquisa Operacional*. Tradução por Helena L. Lemos. Rio de Janeiro: Campos, São Paulo: EDUSP, 1988, 1. ed, 805 p. Tradução de: Introduction to Operations Research, 3. ed.

[100] HO, Y. C., CAO, X. Perturbation Analysis and Optimization of Queueing Networks. *Journal of Optimization Theory and Applications*, USA, v. 40, n. 4, p. 559-82, Aug. 1983

[101] HO, Y. C., CASSANDRAS, C. A New Approach to the Analysis of Discrete Event Dynamic Systems. *Automatica*, Great Britain, v. 19, n. 2, p. 149-67, 1983.

[102] HOBROUGH, P. A. Mainframe or Micro? - A case study. *Journal of Operational Research Society*. Great Britain, v. 14, n. 2, p. 137-44, Feb 1983.

[103] HUANG, Philip Y., CHEN, Chin-Sheng. Flexible Manufactuing Systems: An Overview And Bibliography. *Production and Inventory Management*, USA, v. 27, n. 3, p. 80-90, Third Quarter 1986.

[104] HURRION, R. D. Visual Interactive Modelling. *European Journal of Operational Research*, North-Holland, v. 23, n. 3, p. 281-87, Mar. 1986.

[105] HUTCHINSON, George K. Flexible Manufacturing Systems and Simulation. *Flexible Manufacturing Systems. Edited by John R. Holland and published by SME*, p. 222-23, 1984.

[106] HUTCHINSON, George K. Simulation: An Activity, not a Language. *Proceedings of the Second IFIP WG 5.7 Working Conference on Advances in Production Management Systems--APMS 85*, Hungary, p. 157-64, Aug. 1985.

[107] HUTCHINSON, G. K., WYNNE, B. E. A Flexible Manufacturing System. *Industrial Engineering*. USA, v. 5, n. 12, p. 10-17, Dec 1973.

[108] HWANG, Syming Simon. *Models for Production Planning in Flexible Manufacturing Systems*. USA: University of California, Berkeley, 1986. Ph.D. Thesis.

[109] JOHNSON, M. Eric, POORTE, Jacob P. A hierarchical approach to computer animation in simulation modeling. *Simulation*, USA, v. 50, n. 1, p. 30-6, Jan. 1988.

[110] JONATHAN, Miguel. *Introdução a programação orientada para*

objetos em SMALLTALK. Rio de Janeiro: UFRJ/UCE, 1988, 75 p

[111] JONES Jr, William C. Data Strutures Using Modula-2. New York: John Willey & Sons, Inc., 1988.

[112] KALKUNTE, M. V., SARIN, S. C., WILHELM, W. E. Flexible Manufacturing Systems: A Review of Modeling Approaches for Design, Justification and Operation. *Flexible Manufacturing Systems: Methods and Studies. Edited by Andrew Kusiak*, North-Holland, v. 12, p. 03-25, 1986.

[113] KARIAN, Zaven A. PC Simscript II.5. *Byte*, USA, v. 12, n. 7, p. 244-46, July 1987.

[114] KAUBISCH, W. H., PERROT, R. H., HOARE, A. R. Quasiparallel Programming. *Software-Practice and Experience*, UK, n. 6, p. 341-56, 1976

[115] KIENBAUM, Germano de Souza, BARROS, Maria Suelena Santiago. Modelagem e Simulação Discreta: Novas Metodologias. In: *XX Simpósio Brasileiro de Pesquisa Operacional - XX SBPO*, 4 a 6 de nov. de 1987, Salvador, Bahia. *Anais* ... p. 149-62.

[116] KLAHORST, Tom. *Flexible Manufacturing Systems. Edited by John R. Holland and published by SME.* 1984.

[117] KODATE, H., FUJII, Ken'Ichi, YAMANOI, K. Representation of FMS with Petri Net Graph and its Application to Simulation of System Operation. *Robotics & Computer-Integrated Manufacturing.* v. 3, n. p. 275-83, 1987.

[118] KRIZ, J., SANDMAYR, H. Extension of Pascal by Coroutines and its Apllication to Quasi-parallel Programming and Simulation. *Software-Practice and Experience*, UK, n. 10, p. 773-89, 1980.

[119] KUMARA et al. Expert System in Industrial Engineering. *International Journal of Production Research.* USA, v. 24, n. 5, p. 1107-25, 1986.

[120] KUSIAK, Andrew. Application of Operational Research Models and Techniques in Flexible Manufacturing Systems. *European Journal of Operational Research*, North-Holland, v. 24, n. 3, p. 336-45, Mar. 1986.

[121] KUSIAK, Andrew. Flexible Manufacturing Systems: A Structural Approach. *International Journal of Production Research*, London, v. 23, n. 6, p. 1057-73, July 1985.

[122] KUSIAK, Andrew, HERAGU, Sunderesh S. The Facility Layout Problem. *European Journal of Operational Research.* North--Holland, v. 29, n. 3, p. 229-51, July 1987.

[123] LAUGUERY, Ronald K. Modeling human operators on a micro: A micro version of SAINT. *Simulation*, USA, v. 44, n. 1, p. 10-16, Jan. 1985.

[124] LAW, Averill M., HAIDER, S. Wall. Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey. *Industrial Engineering*, USA, v. 21, n. 5, p. 33-46, May 1989.

[125] LAW, Averill M. Simulation In The Automotive Industry. *Industrial Engineering*. USA, v. 22, n. 11, p. 19-20, Nov. 1990.

[126] LEDBETTER, William N., COX, James F. Are OR Techniques Being Used? *Industrial Engineering*, USA, v. 9, n. 1, p. 19-21, Feb. 1977.

[127] LEE, Jim, CHOI, Richard Hoo-Gon, KHAKSAR, Majid. Evaluation of Automated Guided Vehicle Systems by Simulation. *Computers in Industrial Engineering*, UK, v. 19. n. 1/4, p. 318-21, 1990.

[128] LENZ, John E. MAST: A Simulation Tool for Designing Computerized Metalworking Factories. *Simulation*. USA. v. 40, n. 2, p. 51-58, Feb. 1983.

[129] LINN, Richard J., WYSK, Richard. A Simulation Model for Evaluating Control Algorithms of an Automated Storage/ /Retrieval System. *Proceeding of the 1984 Winter Simulation Conference*, USA, p. 331-39.

[130] LUK, Maria. Hong Kong Air Cargo Terminals To Work In Synch Because of Simulation Applications. *Industrial Engineering*, USA, v. 22, n. 11, p. 42-44, Nov. 1990.

[131] LUKER, P. A., BURNS, A. Program Generator and Generation Software. *The Computer Journal*, Great Britain, v. 29 n. 4, p. 315-21, 1986.

[132] MAIMON, Oded Z., GERSHWIN, Stanley B. Dynamic Schedulling and Routing for Flexible Manufacturing Systems that Have Unreliable Machines. *Operations Research.* USA, v. 36, n. 2, p. 279-92, Mar./Apr.1988.

[133] MAMALIS, A. G., BILALIS, N. G., KONSTANTINIDIS, M. J. On Simulation Modeling for FMS. *Simulation*, USA, v. 48, n. 1, p. 19-23, Jan. 1987.

[134] MATHEWSON, S. C. Simulation Program Generators: Code and Animation on a P.C. *Journal of Operational Research Society*, Great Britain, v. 36, n. 7, p. 583-89, July 1985.

[135] MATHEWSON, S. C. The Application of Program Generator Software and Its Extensions to Discrete Event Simulation Modeling. *IIE Transactions*, USA, v. 16, n. 1, p. 3-18, Mar. 1984.

[136] MATHEWSON, S. C. Simulation Program Generators. *Simulation*. USA, v. 23, n. 6, p. 181-89, Dec. 1974.

[137] MERRIMAN, Michael F. Automated Interactive Simulation Modeling System: AISIM. *The Journal of Systems and Software*, USA, n. 7, p. 61-72, 1987.

[138] MILTENBURG, G. John, KRINSKY, Itzhak. Evaluating Flexible Manufacturing Systems. *IIE Transactions*, USA, v. 19, n. 2, p. 222-233, June 1987.

[139] MINER, Robin J., ROLSTON, Laurie J. MAP/1 Tutorial. *Proceedings of 1984 Winter Simulation Conference*, USA, p. 59-62, 1984.

[140] MISHRA, P. K., PANDEY, P. C. Simulation Studies of Flexible Manufacturing Systems Using Statistical Design of Experiments. *Computers & Industrial Engineering*, Great Britain, v. 16, n. 1, p. 65-74, 1989.

[141] MOERMAN, P. A. Economic Evaluation of Investiments in New Production Technologies. *Engineering Costs and Production Economics*. Netherlands, v. 13, p. 241-62, 1988.

[142] MONAHAN, George E., SMUNT, Timothy L. Optimal Acquisition of Automated Flexible Manufacturing Processes. *Operations Research*. USA, v. 37, n. 2, p. 288-300, Mar/Apr 1989.

[143] MONTAZERI, M., WASSENHOVE, L. N. Van. Analysis of Scheduling Rules for an FMS. *International Journal of Production Research*. London, v. 28, n. 4, p. 785-802, July/Aug 1990.

[144] MONTAZERI, M., GEIDERS, L. F., WASSENHOVE, N. Van. A Modular Simulator for Design, Planning, and Control of Flexible Manufacturing Systems. *The International Journal of Advancing Manufacturing Technology*. USA, v. 3, n. 1, p. 15-32, 1988.

[145] MOURANT, Ronald R. Simulation for the Operations Research Analyst *Impacts of Microcomputers on Operations Research (Proceedings of a Symposium sponsored by The Computer Science Technical Section of the ORSA)*, USA, p. 219-27, Mar. 1985.

[146] MURAMATSU, Rintaro, ISHII, Kazuyoshi, TAKAHASHI, Katsuhiko. Some Ways to Increase Flexibility in Manufacturing

Systems. *International Journal of Production Research*, London, v. 23, n. 4, p. 691-703, July/Aug. 1985.

[147] NANCE, Richard E. Simulation Modeling: Two Perspectives. *IIE Transactions*, USA, v. 16, n. 1, p. 01-34, Mar. 1984.

[148] NASSR, John J. Tecnologia Fundamental para o Estabelecimento de Ambiente CIM. *Boletim Sobracon*, São Paulo. Ano IV, n. 38/39, p. 16-20, 1988.

[149] NEWTON, D. Simulation Model Calculates How Many Automated Guided Vehicles are Needed. *Industrial Engineering*, USA, v. 17, n. 2, p. 68-74, Feb. 1985.

[150] OATES, Wayne J. Manufacturing Modeling Using RESQ. *Proceedings of the 1984 Winter Simulation Conference*, USA, P. 357-59, 1984.

[151] O'DOWD, Mike. MEX, a Programming Language that Supports Discrete Event Simulation. *Microprocessing and Microprogramming*, North-Holland, v. 27, p. 215-20, 1989.

[152] O'GRADY, Peter. Flexible Manufacturing Systems: Present Development and Trends. *Computers in Industry*. USA, v. 12, n. 3, p. 241-51, 1989.

[153] O'KEEFE, Robert M. An Interactive Simulation Description Interpreter. *Computers & Operational Research*, Great Britain, v. 14, n. 4, p. 273-83, 1987.

[154] O'KEEFE, Robert M. Programming Languages, Microcomputers. *Journal of Operational Research Society*, Great Britain, v. 35, n. 7, p. 617-27, July 1984.

[155] O'KEEFE, Robert. Simulation and Expert Systems - A Taxonomy and Some Examples. *Simulation*, USA, v. 46, n. 1, p. 10-16, Jan. 1986.

[156] O'REILLY, Jean J., PRITSKER, A. Allan B., LILEGDON, William R. SLAM II Tutorial. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 17-19, 1984.

[157] ÖREN, Tuncer I., ZIEGLER, Bernard P. Concepts for Advanced Simulation Methodologies. *Simulation*, USA, v. 32, n. 3, p. 69-82, Mar. 1979

[158] PALANISWAMI, S., HASSAN, M. Z. Simulating the Design and Analysis of Quality Control Systems in Manufacturing. *Engineering Costs and Production Economics*, Netherlands, v. 14, p. 217-227, 1988.

[159] PEDGEN, C. Dennis, SHANNON, Robert E., SADOWSKI, Randall P. *Introduction to Simulation Using SIMAN*. New Jersey:

McGraw-Hill, Inc., 1990, 615 p.

[160] PEDGEN, Claude Dennis et al. SLAM: Simulation Language for Alternative Modeling. *Simulation*, USA, v. 33, n. 5, p. 145-57, Nov. 1979.

[161] PERRY, Ronald F., HOOVER, Stewart V., FREEMAN, David R. An Optimum-Seeking Approach to the Design of Automated Storage/Retrieval System. *Proceeding of the 1984 Winter Simulation Conference*, USA, p. 349-54.

[162] PHILLIPS, Don T. Simulation of Material Handling Systems: When and Which Methodology? *Industrial Engineering*, USA, v. 12, n. 9, p. 65-80, Sept. 1980.

[163] PIDD, M. *Computer Simulation in Management Science*. New York: John Wiley & Sons, 1988, 2nd ed., 307 p.

[164] PIDD, Michael (Ed.) *Computer Modelling for Discrete Simulation*. UK: John Wiley & Sons Ltd., 1989. 274 p.

[165] PIMENTEL, Maj Qem M. *Sistema Computacional para simulação discreta com opção para amostragem descritiva*. RJ, Rio de Janeiro: Istituto Militar de Engenharia, 1989. Dissertação de Mestrado.

[166] QUANDT, John A. Implementation of Operational Evaluation Modeling in Pascal. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 763-72.

[167] RACZYNSKY, Stanislaw. Passion - Pascal-Related Simulation Language for Small Systems. *Simulation*, USA, v. 46, n. 6, p. 239-43, June 1986.

[168] RAMACHANDRAN, V., KIMBLER D. L. Economic Comparison of Conventional and Flexible Manufacturing Systems by Simulation. *Computers & Industrial Engineering*, Great Britain, v. 13, n. 1/4, p. 134-37, 1987.

[169] RAMSLI, E. Modelling and Simulation of Manufacturing Systems. *Annals of the CIRP*, Germany, v. 38, n. 1, p. 413-116, 1989.

[170] RANKY, Paul G. *Computer Integrated Manufacturing*. UK: Prentice/Hall International, 1986, 506 p.

[171] RANYARD, J. C. A History of OR and Computing. *Journal of Operational Research Society*. Great Britain, v. 39, n. 12, p. 1073-86, Dec 1988.

[172] RAVINDRAN, A., PHILLIPS, T., SOLBERG, James J. *Operations Research - Principles and Practice*. New York: John Wiley & Sons, 1987.

[173] RO, In-Kyo, KIM, Joong-In. Multi-Criteria Operational Control Rules in Flexible Manufacturing Systems (FMSs). *International Journal of Production Research.* London, v. 28, n. 1, p. 47-63, Jan/Feb 1990.

[174] RUCKER, David. Advanced Factory Simulation Boosts Design, Implementation For GE Aircraft Engines. *Industrial Engineering*, USA. v. 22, n. 4, p. , Apr. 1990.

[175] RUSSELL, Edward C. Simscript II.5 Tutorial. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 43-48, 1984.

[176] SAFIZADEH, M. Hossein. Optimization in Simulation: Current Issues and the Future Outlook. *Naval Research Logistics.* USA, v. 37, n. 6, p. 807-25, Dec. 1990.

[177] SAFIZADEH, M. Hossein, THORNTON, Billy M. Optimization in Simulation Experiments Using Response Surface Methodology. *Computer & Industrial Engineering*, Great Britain, v. 8, n. 1, p. 11-27, 1984.

[178] SALIBY, Eduardo. *Repensando a simulação: A amostragem descritiva* (Coleção Coppead de Administração). São Paulo: Atlas, Rio de Janeiro: Editora da UFRJ, 1989.

[179] SAMMET, Jean E. *Languages: History and Fundamentals.* Englewood Cliffs, New York: Prentice-Hall, Inc., 1969.

[180] SAUL, Greg. Flexible Manufacturing System is CIM at the Shop Floor Level. *Industrial Engineering.* USA, v. 17, n. 6, p.35-39, June 1985.

[181] SAYERS, Ralph D. Simulation Modeling At TRW Provides Insight On Assembly Line Operations. *Industrial Engineering*, USA, v. 21, n. 8, p. 20-24, Aug. 1989.

[182] SCHIMDT, J. W., TAYLOR, R. E. *Simulation and Analysis of Industrial Systems.* Irwin, Homewood Hill, 1970.

[183] SCHRIBER, Thomas J. *Simulation Using GPSS.* New York: John Willey & Sons, 1974. 533 p.

[184] SCHROER, Bernard J., TSENG, Fan T. An Intelligent Assistant for Manufacturing System Simulation. *International Journal of Production Research.* London, v. 27, n. 10, p. 1665-83, Oct 1989.

[185] SCHROER, Bernard J. & TSENG, Fan T. Modelling Complex Manufacturing Systems Using Discrete Event Simulation. *Computers & Industrial Engineering*, Great Britain, v. 14, n. 4, p. 455-64, 1988.

[186] SCOGGINS, Shwu-Yan Chang. *The methodologies of System analysis and Design for Computer Integrated Manufacturing (CIM)*. USA: Texas Tech University, Dec. 1986, Ph.D. Thesis

[187] SEILA, Andrew F. Discrete Event Simulation in Pascal with SIMTOOLS. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 499, 1984.

[188] SEILA, Andrew F. SIMTOOLS: A software tool kit for discrete event simulation im Pascal. *Simulation*, USA. v. 50, n. 3, p 93-9, Mar. 1988.

[189] SELVARAJ, Sathyakumar et al. Discret Event Simulation in C with Disc. *Computers & Industrial Engineering*, Great Britain, v. 18, n. 3, p. 263-74, 1990.

[190] SHANKER, Kripa, TZEN, Ya-Juei Jeffrey. A Loading and Dispatching Problem in a Random Flexible Manufacturing System. *International Journal of Production Research*, London, v. 23, n. 3, p. 579-95, Mar. 1985.

[191] SHANNON, Robert E., BUCKLES, Billy P. Operation Research Methodologies in Industrial Engineering: A Survey. *AIIE Transactions*, USA, v. 12, n. 4, p. 363-67, Dec. 1980.

[192] SHANNON, Robert E. Knowledge Based Simulation Techniques for Manufacturing. *International Journal of Production Research.* London, v. 26, n. 5, p. 953-73, May 1988.

[193] SHANNON, Robert E. Simulation: A Survey with Research Suggestions. *AIIE Transactions*, USA, v. 7, n. 3, p. 289-301, Sept. 1975.

[194] SHARIT, Joseph, SALVENDY, Gavriel. A Real-Time Interactive Computer Model of a Flexible Manufacturing System. *IEE Transactions*. USA, v. 19, n. 2, p. 167-77, June 1987.

[195] SHARMA, Davi, ROSE, Lawrence L. Modular Design for Simulation. *Software-Practice and Experience*, UK, v. 18, n. 10, p. 945-66, Oct. 1988.

[196] SHEPPARD, S. Applying software engineering to simulation. *Simulation*, USA, v. 40, n. 1, p. 13-19, Jan. 1983.

[197] SOLOT, Philippe. A Concept for Planning and Scheduling in an FMS. *European Journal of Operational Research*, North Holland, v. 45, n. 1, p. 85-95, Mar. 1990.

[198] SONNTAG, Victoria. Flexible Manufacturing ... From A Different Perspective. *Industrial Engineering*, USA, v. 22, n. 11, p. 58-61, Nov. 1990.

[199] STANDRIDGE, Charles R. Performing Simulation Projects with

The Extendeded Simulation System (TESS). *Simulation*, USA. v. 47, n. 6, p. 283-91, Dec. 1985.

[200] STANDRIDGE, Charles R., WALKER, Steven A., VAUGHAN, David. A Tutorial on TESS: The Extended Simulation System. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 51-6, 1984.

[201] STECKE, Kathryn E. Formulation and Solution of Nonlinear Integer Production Planning Problems for Flexible Manufacturing Systems. *Management Science*, USA, v. 29, n. 3, p. 273-87, Mar. 1983.

[202] STECKE, Kathryn E. A Hierarchical, Approach to Solving Machine Grouping and Loading Problems of Flexible Manufacturing Systems. *European Journal of Operational Research*, North-Holland, v. 24, n. 3, p. 369-78, Mar. 1986

[203] STECKE, Kathryn E. Useful Models to Adress FMS Operating Problems. *Proceedings of the Second IFIP WG 5.7 Working Conference on Advances in Production Managements Systems - APMS 85*, Hungary, p. 117- 29, Aug. 1985

[204] STECKE, Kathryn E. & KIM, Ilyong. Performance Evaluation for Systems of Pooled Machines of Unequal Sizes: Unbalancing Versus Balancing. *European Journal of Operational Research*, North-Holland, v. 42, n. 1, p. 22-38, Sept. 1989.

[205] STECKE, Kathryn E., SOLBERG, James J. Loading and control policies for a flexible manufacturing system. *International Journal of Production Research*, UK, v. 19, n. 5, p. 481-90, 1981.

[206] STRACK, Jair. *GPSS: Modelagem e Simulação de Sistemas*. Rio de Janeiro: LTC Editora SA, 1984. 174 p.

[207] STYLIANIDES, Chris, RADI, Gade, EBELING, K. A. Use of Simulation In the Analysis of Shop Floor Operations. *Computers & Industrial Engineering*, Great Britain, v. 13, n. 1/4, p. 144-48, 1987.

[208] SUBRAHMANIAN, Eswaran, CANNON, Robert L. A Generator Program for Models of Discret-Event Systems. *Simulation*, USA, v. 36, n. 3, p. 93-101, Mar. 1981

[209] TABIBZDEH, Kambiz. Simulation System for Material Handling System Design. *Computers & Industrial Engineering*, Great Britain, v. 17, n. 1/4, p. 270-73, 1989.

[210] TAHA, Hamdy A. The Simnet Simulation Language. *Computers &*

Industrial, Great Britain, v. 14, n. 3, p. 281-95, 1988.

[211] TALAVAGE, Joseph & HANNAN, Roger G. *Flexible Manufacturing Systens in Practice: Aplications, Design and Simulation.* Series: Manufacturing Engineering and Materials. New York: Dekker, 1988, 358 p.

[212] TCHIJOV, Iouri, SHEININ, Roman. Flexible Manufacturing Systems (FMS): Current Diffusion and Main Advantages. *Technological Forecasting and Social Change*, North Holland, v. 35, n. 2/3, p. 277-93, Apr. 1989.

[213] *The FMS report*, Ingersoll Engineers. Ed. John Mortimer, UK: IFS (Publications) Ltd., 1984, 179 p.

[214] THESEN, A., SUN, Zhanshan, WANG, Tzyh-Jong. Some Efficient Random Number Generators for Micro-Computers. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 187-96, 1984.

[215] THESEN, Arne & SUN, Zhanshan. The Effect of the Choice of Programming Language on the Efficiency of Microcomputer Based Simulation System. *Impacts of Microcomputers on Operations Research (Proceedings of a Symposium sponsored by The Computer Science Technical Section of the ORSA)*, USA, p. 238-48, Mar. 1985.

[216] THOMAS, George & DaCOSTA, Jo-Anne. A Sample Survey of Corporate Operations Research. *Interfaces*, USA, v. 9, n. 4, p. 102-11, Aug. 1979.

[217] TOBIN, Carolyn D. & McCOLLOM, Neal N. A Conceptual Design for Simulation. *Proceedins of the 1984 Winter Sumulation Conference*, USA, p. 621-24.

[218] TOM, Lynette. SEE WHY. *Simulation*, USA. v. 45, n. 2, p. 96, Aug. 1985.

[219] TURBAN, Efrain. A Sample Survey of Operations-Research. Activities at the Corporate Level. *Operations Research*, USA, v. 20, n. 3, p. 708-21, May/June 1972.

[220] UNGER, Brian W. Programming Languages for Computer System Simulation. *Simulation*, USA, v. 30, n. 4, p. 101-10, Apr. 1978.

[221] UYENO, Dean H. & VAESSEN, Willem. PASSIM: A Discrete-Event Simulation Package for PASCAL. *Simulation*, USA, v. 35, n. 6, p. 183-90, Dec. 1980.

[222] VASUDEV, Vinay K., BILES, William E. Microcomputer-based Modeling and Simulation. *European journal of Operational*

Research. North-Holland. v. 24, n. 1, p. 30-36, Jan 1986.

[223] WAGNER, Harvey M. *Pesquisa Operacional.* Rio de Janeiro: Prentice-Hall do Brasil, 1986, 2 ed., 851 p.

[224] WANG, Helen. An Experimental Analysis of the Flexible Manufacturing System (FMS). *Flexible Manufacturing Systems: Methods and Studies. Edited by Andrew Kusiak.* North-Holland, 12: 319-39, 1986.

[225] WARREN, H. J. et al. AUTOSIM: An Automatic Simulation Program Generator. *Mathematics and Computers in Simulation*, North-Holland, v. 27, p. 107-114, 1985.

[226] WELSH, J., BUSTARD, D. W. Pascal-Plus-Another Language for Modular Multiprogramming. *Software-Practice and Experience* UK, n. 9, p. 947-57, 1979.

[227] WHITT, Ward. Planning Queueing Simulations. *Management Science*, USA, v. 35, n. 11, p. 1341-66, Nov. 1989.

[228] WILBANKS, John., STAFFORD, Edward F., SCHROER, Bernard J. Simulation Improves Firm's Competitive Edge and Increases Productivity. *Industrial Engineering*, USA, v. 22, n. 1, p. 32-38, Jan. 1990.

[229] WILSON, James R., PRITSKER, A. Alan B. Computer Simulation. In: *Handbook of Industrial Engineering.* Gavriel Salvendy (editor), USA: John Wiley & Sons, 1982.

[230] WU, Chin-Yun Albert. *A C-Based Interactive FMS decision Support Software for Capacity Planning, Layout Design, and Shop Floor Control.* USA: Case Western Reserve University, Jan. 1987, Ph.D. Thesis.

[231] WU, Szu-Yung David, WYSK, Richard A. An Application of Discrete-Event Simulation to on-Line Control and Scheduling in Flexible Manufacturing. *International Journal of Production Research.* London, v. 27, n. 9, p. 1603-23, Sept. 1989.

[232] WYATT, Dana L., SHEPPARD, Sallie. A Language-Directed Discret Simulation System. *Proceedings of the 1984 Winter Simulation Conference*, USA, p. 463, 1984.

[233] YOUNG, C., GREENE, A. *Flexible Manufacturing Systems.* New York, NY: American Management Association, 1986.

[234] YOUNG, Robert E., ROSSI, Michael A. Toward Knowledge-Based Control of Flexible Manufacturing Systems. *IEE Transactions*, USA, v. 20, n. 1, p. 36-42, Mar. 1988.

[235] ZEIGLER, Bernard P. System-Theoretic Representation of

Simulation Models. *IIE Transactions*, USA, v. 16, n. 1, p. 19-33, Mar. 1984.

[236] ZELENOVIC, D. M. Flexibility - A Condition for Effective Effective Production Systems. *International Journal of Production Research.* v. 20, n. 3, p. 319-37, May/July 1982